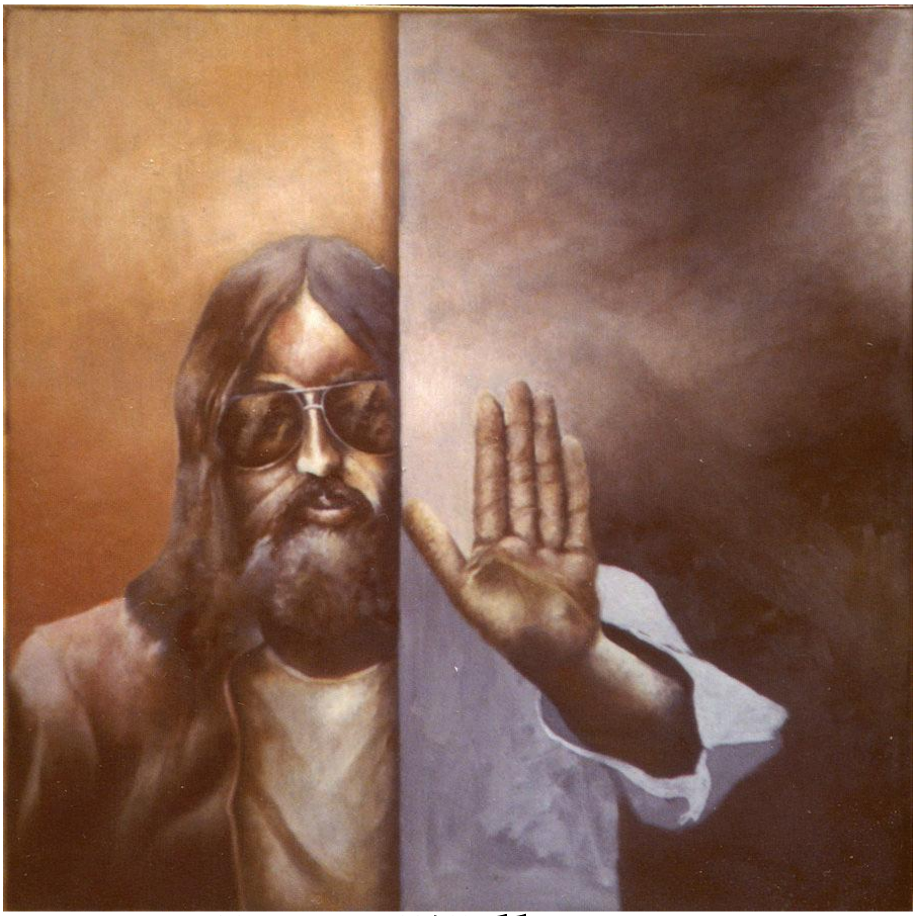
Att använda GNU/Linux



Linus Walleij

Bokens hemsida: `http://www.df.lth.se/~triad/gnulinux/`

Innehåll

KAPITEL 1

Inledning

> När någon säger ”jag vill ha ett operativsystem där jag bara behöver säga vad jag vill ha gjort”, ge honom en slickepinne.
>
> — Okänd

Detta är en bok om operativsystemet GNU/Linux, eller bara Linux, som många kallar det. Redan den plurala namngivningen GNU/Linux, antyder något om hur operativsystemet såväl som detta verk, boken, har strukturerats. Boken speglar systemet.

GNU/Linux-beteckningen förordas av Free Software Foundation av det enkla skälet att de anser att deras arbete med det system som i folkmun bara kallas Linux på detta vis kommit att skuggas av den kärna som skrivits på initiativ av den så omhuldade Linus Torvalds. Inte ens GNU/Linux är egentligen en rättvisande beteckning, men mer fullständiga beteckningar i stil med Qt/KDE/X/GNU/Linux skulle inte precis se vackert ut heller. GNU/Linux är en bra kompromiss som namnger de två viktigaste beståndsdelarna i världens näst vanligaste operativsystem: GNU och Linux. Vad är dessa ting? Det skall boken försöka förklara.

Boken har som mål att presentera GNU/Linux på ett vis som lämpar sig för en användare som vill använda sin egen dator som arbetsstation, och vill förstå vad den gör och hur systemet är uppbyggt. Boken är också tänkt att kunna användas som referensverk för att hantera vissa vanliga problem, eller ge specifika kunskaper på ett visst område.

1.1 Läsanvisning

Resten av detta kapitel är menat som introduktion av vissa grundläggande begrepp. Om du kan det här med datorer så skumma och se att allt är bekant och hoppa till nästa kapitel.

De övriga kapitlen bygger på varandra. Jag introducerar först POSIX-system rent generellt för att sedan begränsa perspektivet till GNU/Linux-system. Om du redan kan något POSIX-system bra, till exempel Solaris, kan du kanske hoppa över detta kapitel, eller nöja dig med att läsa det översiktligt.

Om du precis har skaffat dig en Linux-distribution av något slag, såsom RedHat Linux eller Debian, är det möjligt att du vill installera den så fort som möjligt för att ha något att öva dig på. Du kan i så fall hoppa direkt till kapitel 4 för att hitta det du söker. Detta kapitel har skrivits så att det ska gå att läsa fristående i viss grad. Gå sedan tillbaka och läs boken från början om du vill lära dig mer.

Många av de som börjar använda GNU/Linux kommer från Microsoft Windows-världen. Av detta skäl finns det ett speciellt appendix med namnet *Att byta från Windows till GNU/Linux*, appendix A. Möjligen vill du läsa detta appendix först, det beror på. Det skadar aldrig att smygtitta lite.

Slutligen finns även ett översiktligt appendix som jag kallat för *stordrift av GNU/Linux-system*. Detta är ett vanligt användningsområde för GNU/Linux, men att behandla ämnet uttömmande är inte möjligt — bokens sidor räcker inte till.

I en del kapitel förekommer ord som *foo*, *bar*, *foobar* och *fnord*. Dessa är mer eller mindre vedertagna motsvarigheter till matematikens X och Y, d.v.s. variabler som kan bytas ut mot något annat i en verklig situation.[1]

1.1.1 Andra upplagan

Inför andra upplagan har boken omarbetats och uppdaterats lite här och var. Nya avsnitt har införts om datorns moderkort och trådlösa nätverk. Avsnittet om *X* har uppdaterats för att matcha konsensusbytet till X.org-servern. Andra tillägg har gjorts här och var där det befunnits relevant.

Jag har rättat stora mängder stavfel och säkerligen skapat nya.

[1]Dessa kallas ibland *metasyntaktiska variabler*: *Foobar* kommer möjligen från ordet *fubar* ur amerikansk soldatslang från första (och andra) världskriget (f**ed up beyond all recognition). *Fnord* kommer från den skönlitterära Illuminatustrilogin av Robert Shea och Robert Anton Wilson.

Slutligen har avsnittet med en ekonomisk förklaring till fri mjukvara skrivits om från början med en helt ny utgångspunkt. Den gamla varianten var inte bra, och var säkerligen också orsaken till att någon kallade boken för ”onödigt predikande”. Om boken ändå uppfattas som predikande så är det inte min avsikt: jag vill jämföra operativsystemet GNU/Linux med andra utan fördomar och utan att kora vinnare. Kanske jag ibland kan misslyckas med det.

1.2 Läsarens förkunskaper

Detta är ingen nybörjarbok om att använda datorer. Inte heller är detta en hårdvarubok. Här kommer inte att redogöras för hur du kopplar in sladdarna i din dator och slår på strömbrytaren, vad en mus eller skrivare är till för, eller hur en floppydisk eller CD-ROM är tänkt att användas. Sådant får du lära dig bättre och mer pedagogiskt på annat håll.

Detta är heller ingen ”kom igång med”-bok. Jag räknar med att du är begåvad nog att ta en distribution av GNU/Linux och begripa hur du startar installationen. Emellertid: om det uppstår problem under själva installationen, och du vill lösa detta problem själv, kan denna bok mycket väl innehålla svaret. Det är därför boken har ett ordentligt index, så att du ska kunna hitta de relevanta delarna lätt.

Överlag har jag inte mycket till övers för böcker av typen *”The complete morons guide to running Linux”*. Michi Henning spekulerade vid ett tillfälle[2] i hur världen skulle se ut om det publicerades böcker i stil med *Brain surgery in 14 easy lessons*, *Complete Idiots Guide to Contract Law* etc. Jag vill ta detta tillfälle att provocerande ställa frågan: varför skall datoranvändning vara *enkelt*? Vem har fått för sig detta?

Det vanliga svaret brukar vara att alla skall och måste använda datorer i detta samhälle. Inom religion och andra områden talas det ibland om *exoteriska* läror, som läror för massorna. Inom den gammalgrekiska religionen kunde detta vara mytologin med alla dess mytiska väsen. En folkets religion som inte ställde några svåra frågor. Den *esoteriska* eller ”hemliga” kunskapen, om världens och religionens verkliga väsen, var förbehållen en mindre skara lärjungar. Därmed inte sagt att den skall vara fysiskt svåråtkomlig, kunskapen skulle finnas där för de som aktivt sökte den. Detta är, om du så vill, en esoterisk bok. Det finns inget ”enkelt sätt” att lära sig en sådan sak som matematik, och det är inte enklare med ett operativsystem.[3]

[2] Se [12]

[3] Och trots detta har livet stick i stäv med all logik lärt mig följande: när datorsystem

Det är kanske inte alla förbehållet att förstå och använda GNU/Linux-system på djupet, eller att administrera dem. Detta är alltså i den meningen en elitistisk bok, som riktar sig till en publik med förmåga och vilja att förstå abstrakta resonemang. Den är inte för GNU/Linux-system vad en körskola är för bilar, alltså något alla skall kunna begripa.

Den är en bok om tekniken inuti GNU/Linux. Den är avsedd för den typ av människor som alltid köper en servicemanual till sin bil och inte tvekar att sticka in huvudet under motorhuven. Och precis som är fallet med bilar fordras det inte att du är ingenjör för att klara av saken hjälpligt.

Om du "bara vill använda Linux", eller "bara vill att det ska funka" så sluta läs här och köp en annan bok. Naturligtvis vill jag att alla skall använda GNU/Linux, och nog tror jag att att den här boken bidrar också till det. Min erfarenhet säger mig dock, att bakom varje användare av ett operativsystem, som inte själv kan sköta systemet, står en person som denne ringer till när denne får problem. Ibland är denna person avlönad och heter "helpdesk", och ibland är det en oavlönad kompis, granne eller annan person i bekantskapskretsen. Detta gäller *alla* operativsystem och *alla* användare. Den här boken är för den där personen som användaren ringer till, när problem uppstår.

Jag ska understryka att det definitivt är *värt* att lära sig GNU/Linux på djupet, till exempel till en början genom att läsa denna bok. Det är först med förståelse för de tekniska detaljerna och systemets grundtankar, som du till fullo kan förstå vidden av och skönheten i GNU/Linux och andra POSIX-system.

En del av de som kommer i kontakt med denna bok kommer säkert att tycka att den är alldeles *för enkel*. Det är bra för er! Ge boken till nån som behöver den istället.

1.3 Om formen

Jag har envisats med att i denna bok konsekvent kalla operativsystemet som avhandlas för *GNU/Linux*, medan dess kärna kallas *Linux*.

En kommentar jag hört om detta är, något i stil med: *Alla kallar det ju bara för Linux, det är meningslöst att försöka definiera mer precisa begrepp, även om det är korrekt, för ingen fattar det.*

Förutom att jag tror detta vore felaktigt och föraktfullt mot bokens förhoppningsvis självständigt tänkande läsare, så är min vision av mitt eget författarskap att detta handlar om att lyfta andra till samma nivå

skall förklaras är det vissa människor som anser att det låter obskyrt, är hopplöst krångligt, och som inte sällan ger uttryck för en viss arrogans och ointresse. Jag undrar om de skulle behandla utlåtanden från sin läkare eller advokat på det viset.

som jag själv, inte att jag skall sänka mig till några ospecificerade ”andras” låga nivå. Ett liknande resonemang gäller valet av begreppet *fri mjukvara* istället för den populära termen *öppen källkod* (engelska: Open Source). Den senare är något jag uppfattar som mest förvirrande.

Denna bok innehåller ingen ordlista. Den har däremot ett omfattande index där du kan hitta de ställen där orden är definierade i sitt rätta sammanhang, på så vis läser du lite mer och lär dig lite mer än du skulle gjort om du bara fått definitionen på ett ord.

När det dyker upp en fotnot i texten[4] betyder detta att det i fotnoten finns mer utbroderande anmärkningar som skulle kunna verka onödigt förvirrande vid en första genomläsning. Om du just har läst något som verkade intressant och stöter på en fotnot, bör du läsa fotnoten för mer information. Fotnoter används ibland även för utvikande förklaringar om något är lite oklart, t.ex. om jag använder ett begrepp som kanske inte är allmänt vedertaget, samt för hänvisningar till vissa källor.

En litteraturhänvisning av det här slaget[30] anger en källa där du kan hitta de originaltexter som citeras, eller läsa mer om något specifikt ämne. En litteraturlista som svarar mot dessa nummer återfinns i slutet av boken, före indexet.

1.4 Bäst-före-datum

När jag sist gav mig tid att skriva en fackbok som berörde tekniska områden fick jag på ett direkt vis erfara att litteratur av detta slag har en synnerligen kort halveringstid.

På samma vis kan denna text komma att revideras, beroende på tid och möjlighet. Vad beträffar referenserna till Internet sist i boken är dessa väl närmast att betrakta som ett skämt, men den som har tillgång till ett arkiv med Internetfiler av typen *The Wayback Machine*[5] kan kanske kolla vad som fanns på de angivna URL:erna vid angivet datum. Internetadresser som anges direkt i texten eller i fotnoter är naturligtvis inte heller inhuggna i sten.

1.5 Datorvetenskaperna

Nu några ord om de fackmänniskor som eventuellt skulle kunna ha nytta av denna bok.

---

[4] D.v.s. en sådan här

[5] Se: `http://www.archive.org/`

Det finns minst fem akademiska discipliner som sysslar med datorer. Att känna till dessa och deras inriktning och dess anhängares egenheter är nödvändigt för att kunna bedöma värdet och relevansen av den information de producerar:

Datalogi är läran om datorns teoretiska aspekter, i synnerhet programmering och programvarukonstruktion. Sorteringsalgoritmer, operativsystem, artificiell intelligens och diverse matematisk exercis som relaterar direkt till datorer och inte primärt till matematik är populärt bland anhängare av denna disciplin. Favoritapplikation: graftraversering på polynomiell tid. Förebild: Donald Knuth.

Datorkommunikation är läran om hur datorer kommunicerar i nätverk. Detta område behandlar köteori, nättopologi, datorsäkerhet, protokoll o.s.v. Företrädarna älskar routrar, twisted-pair-kabel, analyser av DoS-attacker,[6] buffertöverskrivningar och allt annat som gör livet värt att leva. Favoritapplikation: webbserver. Förebild: Agner Krarup Erlang.

Datorteknik är läran om hur datorer konstrueras rent elektroniskt. Vilka processorer och minnen du skall välja, hur de skall sättas samman, hur du tillverkar en egen ASIC eller FPGA o.s.v. är kära ämnen för de som studerar denna disciplin. De flesta är datoringenjörer. Favoritapplikation: styrsystem på kisel. Förebild: Charles Babbage.

Numerisk analys är en matematisk disciplin för vars skull de första datorerna byggdes på 1950-talet. Företrädarna sysslar med att lösa matematiska problem och framställa approximativa lösningar till matematiska problem med hjälp av datorer. Favoritapplikation: iterativ lösning av differentialekvation. Förebild: Isaac Newton.

Systemvetenskap (kallas även **informatik**) går ut på att bygga *system* och när jag ställt frågan till de som arbetar inom detta område har de till och med satt ett frågetecken efter påståendet att de system en systemvetare arbetar med måste innehålla en dator. Området förknippas med förkortningen ADB vilket utläses ”Automatisk Databehandling”. System av den typ som systemvetare gärna arbetar med tenderar att inbegripa framför allt databaser, arbetsflöden och dokumenthantering. Deras huvudsakliga verksamhet är att automatisera ekonomiska och administrativa stödsystem.

[6] DoS, *Denial of Service*, är avsiktliga överbelastningsattacker riktade mot olika former av nätverkstjänster.

> Genom användning av datorer blir detta enklare än det var på den tiden då kartotek fick användas. Favoritapplikation: ”stockticker” (en rullande list som visar börskurser). Förebild: Bill Gates.

Förhoppningen är att boken skall kunna förstås och användas till dagligdags av lärjungar till alla fem disciplinerna, och är ni inte akademiker så kan ni säkert ändå känna en dragning till någon av ovanstående beskrivningar. Systemvetarna är väl allt som oftast de flitigaste användarna av operativsystem, liksom datoringenjörerna, som läser en mix av alla de här ämnena. Det praktiska användandet av ett operativsystem, vilket denna bok handlar om, faller på ett märkligt vis mellan dessa vetenskaper. Traditionellt sett är praktiskt användande av teknik en yrkesutbildning, och anses kanske inte fint nog att lära ut vid universitet, vad vet jag. Kanske är det även bäst så.

1.6 Något grundläggande om operativsystem

För att föra en diskussion runt något fordras att samtalsparterna i någon mån talar samma språk. För detta syfte presenteras här en snabbkurs, eller som anglosaxerna säger, *crash course* i grundläggande operativsystemteori. Detta är dock ingen bok om operativsystem i största allmänhet, och innehållet är därför sparsmakat; den som önskar exakt kunskap i detta område rekommenderas att läsa någon av de utmärkta böcker i ämnet som producerats.[7]

De första datorer som människan skapade saknade operativsystem. I begynnelsen skrevs ett program genom att byta plats på kablar i Z3 som tillverkades i Nazityskland år 1941, den första generellt programmerbara dator som någonsin skapats. Innan dess fanns det specialdatorer, som bara vara byggda för att utföra en enda sak, ibland var de inte ens *digitala* — siffermaskiner — utan analogimaskiner som representerade tal med strömmar, spänningar, vattennivåer eller pneumatiskt tryck. Dessa var avledningar av de första reglertekniska systemen.

Efter Z3 kom ENIAC, den första helt elektroniska datorn, som färdigställdes i USA år 1947. Den var byggd av elektronrör och programmerades också enbart med kablar. Efter den kom sedan datorer, oftast under namnet *matematikmaskiner* att uppstå i en strid ström. Programmeringssystemen utvecklades så att programmen kunde matas in på hålkort istället för med kablar, och resultaten kom ut på hålkort de också. Beräkningsförloppet kunde följas på lampor.

[7] Se exempelvis Abraham Silberschatz bok *Operating System Concepts*[30].

På 1960-talet utvecklades de första operativsystem som kallades operativsystem, bland dem OS/360 från IBM och snart därefter MULTICS[8] som var ett forskningsprojekt som bedrevs av MIT, AT&T Bell Labs och General Electric. Dessa operativsystem var utformade för datorer som hade skärm och tangentbord i form av en enklare terminalskrivmaskin, och ett kunde interagera med användaren via skrivna kommandon.

När hemdatorerna (ibland kallade *mikrodatorer*) dök upp under 1970- och 1980-talen, saknade de i princip operativsystem och hade därför "backat" i datorernas evolutionära utveckling. Det vanligaste var istället att när en hemdator slogs på startades en interaktiv tolk för programspråket BASIC. De första versionerna av IBM PC hade exempelvis denna egenhet, om användaren inte hade stoppat in en diskett med ett operativsystem i datorn.

Därefter fick hemdatorerna också operativsystem: bit för bit byggdes de på med operativsystem, så att de blev som "riktiga" datorer.

Vad menar vi egentligen med ett *operativsystem*?

Den saken är inte helt klar. En av de bästa förklaringar jag känner till är att tänka evolutionärt: först får program skrivas för en viss dator, på ett vis som är specifikt för just den datorn. Inga delar av dessa program kan återanvändas på någon annan dator. Efter hand som datorn används uppstår behov av att återvända vissa ständigt återkommande funktioner, så att hjulet inte behöver uppfinnas på nytt för varje program.

För Datasaab D21, en hiskelig rad av kylskåp som under en tid tillverkades i den stora byggnad som ligger snett bakom Linköpings järnvägsstation, fanns exempelvis programbibliotek med matematiska funktioner (såsom matrismultiplikationer och andra operationer besläktade med lineär algebra) på stora rullband som kunde användas istället för att tillverka helt nya program varje gång.

De återanvändbara delarna får till sist en sammanhållen struktur och kallas då för ett operativsystem, och alla som använder datorn förutsätter att det finns där.

På så vis är ett modernt operativsystem summan av de saker som har befunnits vara bra att återanvända. Användargränssnittet, rutiner för att läsa in data från tangentbord och hårddiskar, rutiner för att skriva ut och lagra data, rutiner för att kommunicera med datornätverk. Allt detta samlas i operativsystemet. De nya datorprogram som skrivs kan sedan räkna med att operativsystemet redan finns där i någon form.

De absolut mest grundläggande delarna av operativsystemet bru-

[8]Multiplexed Information and Computing Service var tänkt att leverera datorkraft till företag och hushåll på samma vis som elektricitet och vatten.



Figur 1.1: En hålremsa med en s.k. bootstrap-loader till en gammal dator av märket Hewlett-Packard.

kar vara inbyggda i datorn i form av ett ROM[9] vid leverans och kallas då ibland för *BIOS* efter engelskans *Basic Input/Output System*. Dessa har rollen av att starta upp och initialisera hårdvaran och sedan ladda in det egentliga operativsystemet. Den specifika funktion som laddar in resten av operativsystemet kallas då ofta *bootstrap loader*, eller kort *boot loader*. Detta namn kommer från tanken på att en dator s.a.s. ”lyfter sig själv i håret” när operativsystemet startas — egentligen behövs nämligen ett operativsystem för att kunna starta ett operativsystem, men bootstrap:en är ett undantag som löser upp detta moment 22.

En bootstrap är en sådan där ögla som sitter baktill på en stövel (engelska: *boot*) och sinnebilden här är alltså en person som lyfter upp sig själv genom att ta tag i denna ögla och dra, något som den legendariske Baron von Münchhausen sade sig kunna göra.

Någon form av minimal BIOS finns nästan alltid, ett äldre sådant kan beskådas i figur 1.1. Programmet på bilden laddas in i datorn och startas, varefter datorn är redo att läsa datorprogram från andra hålremsor och magnetskivor. I senare tid används begreppet boot loader även om ett litet program som ligger först på datorns hårddisk och som väljer och laddar in ett operativsystem som finns på denna hårddisk. Vi kommer senare att stifta bekantskap med två sådana program: LILO (Linux Loader) och GNU GRUB (GNU GRand Unified Bootloader).[10]

---

[9] ROM är en engelsk akronym som utläses *Read Only Memory*, ett halvledarminne som bara kan läsas, ej skrivas.

[10] Detta avhandlas på sidan 138.

1.6.1 Konstruktionshierarkier

> Det som är ovan springer ur det som är inunder, och det som är inunder springer ur det som är ovan, men allt utgår från Det Endas mirakel.[11]
>
> — Hermes Trismegistos smaragdtavla

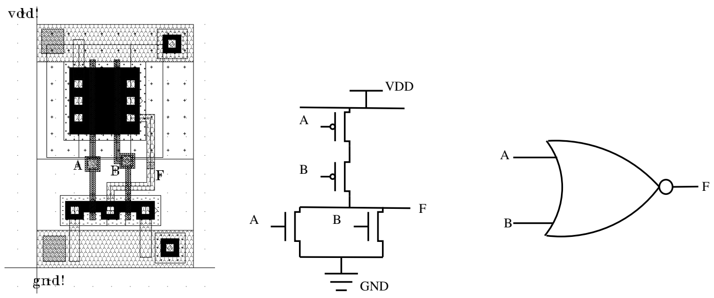
Inom alla tekniska vetenskaper konstrueras system enligt hierarkiska modeller av komponenter, vare sig de är fysiska och tillverkade av materia, eller de är virtuella och manifesterar sig i den fysiska världen bara på det mikroskopiska planet.

Detta hierarkiska komponenttänkande manifesterar sig i sprängskisser över tekniska apparater där barn kan identifiera beståndsdelarna, eller ingenjörers ritningar med noggrann dokumentation av alla ingående detaljer var för sig och lika noggranna instruktioner för hur de skall sättas samman. Vi lär oss tidigt detta grundläggande förhållande till teknik, så grundläggande skulle jag gissa, att den som inte greppar just detta förmodligen aldrig kan förstå teknik ordentligt. I min barndom var det en gul bok i stort format med titeln *Så funkar det* av Joe Kauffman som lärde mig teknikens inre logik.

Komponenttänkandet är första nyckeln till att förstå all teknik. Komponenttänkandet är *hierarkiskt* på så vis att mindre komponenter bildar större komponenter. Den fysiska datorn består av en låda av metall eller plast, med ett eller flera kretskort inuti, anslutna till omvärlden via kontakter. Kretskorten består av elektroniska komponenter och ledningstrådar. De elektriska komponenterna består av olika kemiska föreningar, ledningsbanorna av koppar. En digitalelektronisk komponent består av anslutningspinnar, en plastkapsel och en kiselbricka. Kiselbrickan är fotografiskt etsad med dopade halvledarmönster som utgör logiska grindar. De dopade områdena består av avsiktliga föroreningar av olika grundämnen. Och så vidare, ända ner till subatomära nivåer som vi inte fullt förstår oss på än.

På samma vis kan den logiska konstruktionen i ett operativsystem delas ned i en rad hierarkiska nivåer. Det är en vanlig missuppfattning att tekniska system består av en *strikt* hierarki — detta är felaktigt — de närmaste nivåerna interagerar på ett intrikat vis, och även avlägsna nivåer påverkar varandra. På så vis är ett operativsystem mer som en levande kropp där de olika organen påverkar varandra och bara helhetens harmoni räknas.

[11] Min översättning av den engelska översättningen av den latinska översättningen av den arabiska källan. (Tillhör den alkemiska traditionen.)



Figur 1.2: Här syns hur ett datorchip konstrueras från byggelement till mer abstrakta hierarkier, längst till vänster en förenkling av det faktiska fysiska kretsmönstret på kiselbrickan, som formar de fyra komplementära CMOS-transistorerna. Längst till höger den logiska schemasymbol som symboliserar hela detta kretsmönster.

Den andra nyckeln för att förstå tekniken i ett operativsystem är att det är indelat i *skikt*. Detta innebär att komponenterna är ordnade över eller under varandra med hänsyn tagen till viss funktionalitet. Detta hänger starkt samman med komponenttänkandet, ty varje komponent kan täcka en eller flera nivåer av funktionalitet, men sällan en och en halv nivå. Vanligtvis täcker den bara en enda. I figur 6.3 på sidan 228 ser du till exempel hur många skikt av abstraktioner som döljer sig mellan användargränssnittet i ett GNU/Linux-system och den hårdvara som faktiskt till sist gör jobbet. Grundtanken är följande:

Nivå 1
⇕
Nivå 2
⇕
Nivå 3

En sak som befinner sig på *nivå 1* skall i idealfallet inte behöva bekymra sig om vad som finns på *nivå 3*, utan skall bara utformas på så vis att *nivå 2* kan kommunicera med den. Detta ger en hierarkisk åtskillnad mellan två nivåer av abstraktion. Det är underförstått att det som finns på en lägre nivå är *mer detaljerat* — inte nödvändigtvis *mer komplicerat* eller *viktigare* — än det som finns på en högre nivå. Saker som finns på en högre nivå är *ackumulerade* och *överordnade*, men måste fylla upp krav från underliggande nivåer.

Om vi till detta tankesätt lägger anmärkningen att ett system på en viss nivå följer sin egen dynamik, och inte bara kan förstås i termer av

de underliggande komponenter det består av, kallas den resulterande synen för *holism*. Tanken att en helhet *kan* beskrivas bara som summan av ett antal komponenter skulle däremot kunna kallas för *atomism*. Att intuitivt förstå både delarna och helheten är kärnan för all förståelse av teknologi.

Det är nödvändigt att ha en sådan syn för att förstå datorer. Huruvida en sådan världssyn är tillämplig inom naturvetenskapen i stort — där t.ex. kvarkar bygger atomer, som bygger molekyler, som bygger celler, som bygger organ, som bygger levande varelser — är en annan fråga. Datorsystem är medvetet konstruerade av komponenter, som lagts samman så att de bildar en ny, självständig enhet. De är alltså konstruerade på ett holistiskt vis, och skall förstås holistiskt.[12]

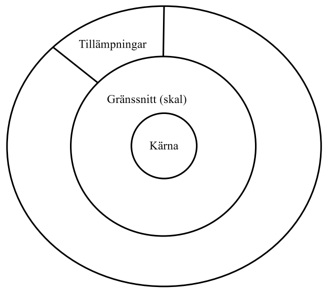
Det *finns* andra sätt att förstå datorer. Ett vanligt är *ceremoniellt* och liknar ett religion: användaren lär sig att upprepa ett visst beteende, som denne varken förstår eller kan förklara. Användaren har lappar med steg-för-steg-beskrivningar för att utföra varje liten uppgift på sin dator. Användaren har ingen mental modell för hur det som finns inuti datorn är uppbyggt utan ser den som en ”svart låda”. Det här beteendet för tanken till magi, och förklarar varför utomstående ibland kan uppfatta en datorteknikers arbete som ren magi: det ser nämligen bara ut som en lång rad obegripliga manövrar eftersom denne saknar en tillämplig förklaringsmodell.

GNU/Linux är medvetet uppbyggt på ett vis som uppmuntrar och möjliggör en djup förståelse för hur systemet är sammansatt och hur de olika delarna samverkar. Lär dig tänka i termer av hierarkiska nivåer, och du lär dig förstå datorer.

1.6.2 Operativsystemets hierarkier

Med ordet *system* i *operativsystem* ligger underförstått att det är ett *system av komponenter*. Om det inte var på det viset skulle det knappast kunna kallas för ett system. Då skulle vi kanske kalla det för *operativmonolit*, *operativatom* eller något liknande som ger uttryck för att det är fråga om en odelbar enhet. Alla någorlunda moderna operativsystem

[12]Nå, det *har funnits* eller *finns i mindre grad* datorsystem som inte konstruerats på detta vis. Operativsystem som skrevs för datorer med trumminnen kunde t.ex. optimeras för att synkroniseras med trummans rotationshastighet. Här förekom inte ens någon avskiljning mellan maskin och program — då är det inte fråga om att de här två nivåerna (maskinen och datorprogrammet) kan uppfattas som system som skall betraktas oberoende av varandra längre, eftersom det ena så klart bär spår av det andra. Det finns också datorprogram av allehanda slag som genom brist på struktur fått en karaktär som knappast kan kallas holistisk. X.org-servern (se avsnitt 6 på sidan 221) brukade tidigare framhållas som ett sådant.

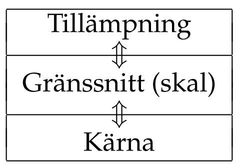


Figur 1.3: Kärnan i operativsystemet.

består av komponenter; hundratals, tusentals eller tiotusentals beroende på hur grovt du sållar.

I en dator kan den innersta delen av operativsystemet, den viktigaste *komponenten* illustreras i figur 1.3. Här finns en *kärna* i centrum, runt vilken flera lager av nötmassa lindat sig.

I ett GNU/Linux-system utgörs kärnan bara av en enda komponent; dess kärna är vad som kallas *monolitisk*. Emellertid visar det sig, om vi tittar närmare på kärnan, att den faktiskt består av *moduler*, d.v.s. komponenter, den också.[13] I Intel 80386-sammanhang talas om olika *ringar*,[14] där den innersta är ring 0, sedan numreras de utåt. Jag kunde lika gärna ha ritat figuren som en holistisk lagerkaka, där varje lager kan begripas i sina egna termer:



Alla saker i figuren är vitala för att just det här operativsystemet på just den här datorn skall fungera utan problem. Hierarkierna i datorn utgår från operativsystemets kärna, vilken startas av datorns *bootstrap loader* då strömmen slås på.

Runt denna kärna och dess gränssnitt (även kallat *skal*) finns en stor mängd tillämpningsprogram. Dessa kan vara av alla möjliga slag, men vissa är så pass grundläggande att de får anses vara en del av själva

[13]Se avsnitt 5.4 på sidan 179.

[14]Intel 80386 (och efterföljare som 80486, Pentium, Pentium II etc.) har fyra olika ”ringar”, med nummer 0, 1, 2 och 3. De flesta operativsystem, däribland GNU/Linux, använder bara ring 0 (”kernelspace”) och ring 3 (”userspace”).

operativsystemet. I någon mån är det en definitionsfråga vad som är en del av operativsystemet och inte, men exempelvis ett kommando för att lista filer såsom POSIX-kommandot **ls** eller det enkla programspråket **awk** är så grundläggande att de får anses vara en del av operativsystemet.

De olika delarna i operativsystemet startas i en sekvens: först aktiveras som sagt BIOS och dess bootstrap loader, därefter laddas operativsystemets kärna. Denna laddar i sin tur in en mängd drivrutiner för att kunna använda datorns hårdvara. Varje individuell drivrutins uppgift är att hantera en viss specifik hårdvara. Drivrutinerna kan ibland identifieras automatiskt, vilket i PC-världen är identiskt med det som kallas *Plug'n'Play*. De kan också laddas in senare under körning, något som kallas *hotplugging*, till exempel då USB-enheter ansluts till datorn under körning. Kärnan kommer då i idealfallet att identifiera enheten och ladda in rätt drivrutin. Är den ännu mer listig försvinner drivrutinen dessutom ur datorns minne då sladden till USB-enheten kopplas ur.

Efter att kärnan laddat drivrutinerna för hårdvaran startas tjänster av olika slag. Dessa varierar mellan operativsystem. I Linux och andra liknande operativsystem används något som kallas *körnivåer* (engelska: *runlevel*) för att kategorisera i vilken nivå, räknat från det mest avskalade operativsystemet, som tjänsterna skall startas. I avsnitt 5.4.2 på sidan 184 kommer vi att gå igenom i detalj hur detta fungerar i GNU/Linux.

Det som presenteras i denna bok är *användarperspektivet* för ett GNU/Linux-system: hur du kan installera, underhålla och använda GNU/Linux för ditt dagliga arbete. Utgångspunkten är ett GNU/Linux-system installerat på en IBM PC-kompatibel dator, men största delen av beskrivningarna gäller även även för GNU/Linux på andra datorer.

Om du går och köper en bok i *operativsystemteori* kommer den inte att handla speciellt mycket om dessa saker. Dessa utgår från *konstruktörsperspektivet* och handlar därför oftast om de allra innersta, mest svårprogrammerade delarna av själva kärnan: hur processer och trådar schemaläggs, hur minne allokeras och deallokeras samt hur filsystemet och andra in/ut-enheter hanteras. I detta fall rör sig litteraturen på en teoretisk nivå utan några som helst exempel på hur en användare upplever det hela och kan kontrollera sina processer, filsystem o.s.v. Denna bok handlar mer om det senare.

1.7 Tack

Flera personer har hjälpt mig att förverkliga den här boken. Flera av dem förtjänar ett särskilt tack för hjälpen.

Först vill jag tacka Ola Larsmo, som genom sina frågor gjorde mig uppmärksam på bristen av användbar litteratur på det här området. Ett speciellt tack riktas till Wilhelm Assarsson som outtröttligt lusläst alla mina manuskript.

Andra som gett värdefulla bidrag genom provläsning är Joachim Strömbergsson och Håkan Kvist, som även haft vänligheten att köra texten igenom *aspell*. Alla fel som finns kvar i texten är naturligtvis mina egna.

Jag vill också tacka alla de personer som gjort stora insatser för fri mjukvara i Sverige: särskilt Mikael Pawlo och Patrik Wallström som drev nyhetsportalen Gnuheter och Jonas Bosson som outtröttligt arbetat med mjukvarupatentfrågan.

De övriga viktiga aktörerna är förhoppningsvis redan uppräknade någonstans i denna framställning, både företag och enskilda personer, men jag tackar er alla, ni vet själva vilka ni är. Tack speciellt till David Weinehall och Christian Rose som jag haft förmånen att tala närmare med om deras projekt.

Tack även till Datorföreningen vid Lunds Universitet och Lunds Tekniska Högskola som är den institution som betytt mest för min skolning i POSIX-system. Det är omöjligt att inte samtidigt också tacka vännerna i SSLUG (Skåne-Själland Linux Users Group) för alla föredrag och installfester.

Ett speciellt tack till de inspirerande personer som alltid svarat på frågor eller kommit med synpunkter: Fredrik Roubert, Peter Svensson, Tomas Gradin, Linus Åkesson, Martin Wahlén, Christian Kullander, Mikael Abrahamsson, Bertil Hedåker och Sven Mattisson. Och tack till alla andra studie- och arbetskamrater överallt.

Tack till Eva och Ivar för allt.

KAPITEL 2

POSIX

Med tanke på datoranvändares och operativsystemanvändares önskningar i största allmänhet, är klagandet över brist på standard den kanske vanligaste litania en administratör eller annan ”datorperson” får höra. Vad få människor tycks känna till är att det faktiskt *existerar* en standard för operativsystem, och denna har namnet POSIX.[1]

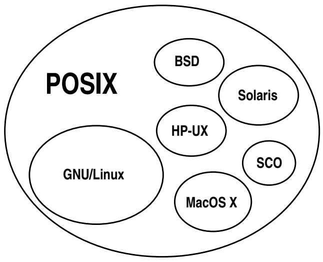
POSIX — vilket skall utläsas Portable Operating System Interface for UNIX — definierar ett abstrakt operativsystem med alla dess grundläggande komponenter och är en så kallad *de jure*-standard från det amerikanska standardiseringsorganet IEEE[2] med numret IEEE 1003.1. Internationellt har denna standard även numret ISO/IEC 9945. Namnet POSIX var ett förslag från programmeraren och initiativtagaren till GNU-projektet, Richard Stallman.

När vi talar om ”UNIX-system” är det oftast i själva verket POSIX-system, system som följer POSIX-standarden, vi menar.[3] En annan beteckning är *UNIX-liknande system* (engelska: *UNIX-like system*). Avsikten med detta kapitel är att ge dig insikter i saker som är gemensamma för

[1]Detta är aldrig fel att påpeka när någon säger att de vill ha installerat ”ett standardiserat operativsystem”. Det givna svaret på frågan är då: ”jag antar att du menar att du vill installera ett system som uppfyller IEEE 1003.1, även känt som POSIX?”. 99 av 100 menar naturligtvis att de vill ha ”samma operativsystem som alla andra använder”, vilket inte är samma sak som ett standardiserat operativsystem.

[2]Institute of Electrical and Electronics Engineers

[3]BSD och GNU/Linux är två av de få operativsystem som faktiskt kan hävdas följa POSIX.



Figur 2.1: Illustration av några av de olika operativsystemen i POSIX-familjen

alla POSIX-system, inte bara GNU/Linux. Kunskaperna här kan tillämpas i nästan alla POSIX-system du stöter på. Det är det absolut viktigaste kapitlet för den som aldrig använt ett POSIX-operativsystem innan.

POSIX är en svåråtkomlig och dyr standardspecifikation som ytterst få personer verkligen har läst.[4] Utöver detta är den utformad för programmerare och tekniker som redan är på det klara med vad som skall avhandlas. Här ges förhoppningsvis en mer lättfattlig beskrivning.

Numera används för övrigt mestadels en nära släkting till POSIX med namnet Single UNIX Specification, och anledningen är sammanfattningsvis dels att vissa saker som saknades i POSIX lagts till i Single UNIX Specification, och att IEEE hårdnackat har vägrat att publicera POSIX-dokumentationen på World Wide Web, där utvecklare av operativsystem som GNU/Linux kan komma åt den. Såväl POSIX-dokumenten som Single UNIX Specification är nyttig läsning och som användare bör du åtminstone ha tittat på dem, se [14][31].[5]

De olika kända operativsystem, som i dagligt tal kallas "UNIX-system" är alla delar i familjen av POSIX-system så som visas i figur 2.1. Anledningen till att de inte skall kallas "UNIX-system" är att UNIX är ett registrerat varumärke som ägs av The Open Group. Det är förbjudet att tillverka och marknadsföra något med namnet UNIX utan licens och certifikation från The Open Group, och det är också därför Linux måste kallas Linux och inte får heta "Linus UNIX" eller något liknande. Ytterst få system har passerat The Open Groups certifieringstest för UNIX, i dagsläget bara Solaris, ett par varianter av AIX och Tru64, och dessa förvisso bara på vissa specifika hårdvaror. Att så många system

[4]Den 3 Juli 1991 efterfrågade Linus Torvalds POSIX-specifikationerna på comp.os.minix men hade inte råd att köpa dem. Istället använde han manualsidorna från SunOS, nuvarande Solaris, som dokumentation av POSIX-gränssnittet.

[5]För mer historia kring standardisering, se Eric S. Raymonds *The Art of Unix Programming*[26].

i allmänhet, och Linux-varianterna i synnerhet inte passerat testerna är mest en kostnads- och tidsfråga: för det första att det kostar. Massor. För det andra tar det tid eftersom certifieringen är en noggrann och långsam process och måste göras för varje ny distribution (se kapitel 4 för detta begrepp), och Linux-distributionerna kommer ut i så snabb takt, att innan en distribution blivit certifierad kan det ha kommit två nya versioner av distributionen, och då är det hela inte särskilt meningsfullt.

Det kan sägas att Linux definitivt är en predikant i POSIX-kyrkan, men saknar den helt onödiga förgyllda biskopskräklan i massivt guld med inskriptionen "UNIX" för att visa sin tillhörighet till denna smala sammanslutning. Många hävdar dessutom att begreppet UNIX är så utbrett att det är orätt och felaktigt av The Open Group att sitta på detta varumärke. Gruppen har på senare tid börjat ta vissa steg mot att förändra sin inställning, en process som sannolikt inte kommer att gå särskilt snabbt.

2.1 Historia

POSIX-systemens historia[6] började år 1969 då Ken Thompson skrev det första UNIX-systemet direkt i assembler för en dator av typen PDP-7 från företaget Digital. Namnet UNIX är en travesti på MULTICS, ett annat tidigt fleranvändarsystem. UNIX var i början ett enanvändarsystem som Thompson använde till att bearbeta text och skriva spel på.[7] Något år senare fick Bell Labs en PDP-11, och tillsammans med kollegan Dennis Ritchie skrev Thompson om hela UNIX från början för denna maskin.

Forskningsdirektörerna på Bell Labs imponerades av Thompsons och Ritchies arbete, och efter att Ritchie skapat programspråket C för att göra operativsystemet portabelt, skrev de om UNIX en tredje gång, nu i C. Det är ingen slump att C har kallats "en stor makroassembler", det var nämligen precis vad det var avsett att vara — ett portabelt assemblerspråk som kunde implementeras på flera datorer så att konstruktörerna slapp skriva om UNIX från början för varje nytt system

[6]Här presenteras en mycket kortfattad variant av UNIX historia, för en mer utförlig och detaljerad historieskrivning, se standardverket *A Quarter Century of Unix* [29].

[7]Numera är POSIX-familjen naturligtvis fleranvändarsystem, hundratals personer kan använda ett och samma GNU/Linux-system samtidigt, även om de inte fysiskt kan sitta vid *konsollen*, den specifika datorns skärm och tangentbord, samtidigt. Fleranvändaregenskapen i POSIX-system upplevs genom att användaren öppnar fönster in i en annan dator och startar sina program där, medan det synliga gränssnittet, oavsett om det är text eller fullgrafiskt, åskådliggörs på den maskin, ofta kallad *terminal* eller *klient* där användaren själv sitter.

som skulle stödjas.[8]

UNIX kom tidigt att erbjudas till flera universitet, och UNIX-användarna framför alla andra kom att bli Berkeley-universitetet[9] strax utanför San Francisco. UNIX levererades till Berkeley i form av källkod på magnetband. Universitetet fick själva se till att kompilera operativsystemet och få det att fungera. Efter att den sjunde versionen av UNIX, känd som Seventh Edition, färdigställts slutade AT&T emellertid att leverera källkod till sitt operativsystem. Universitetet fortsatte då att utveckla operativsystemet för sina egna behov på egen hand, och denna version av UNIX finns kvar ännu idag under namnet BSD — Berkeley Standard Distribution, allmänt kallad ”Berkeley UNIX”. Denna blev under denna tid så populär att universitetet kunde starta en egen avdelning i det enda syftet att vidareutveckla och sälja BSD. Denna började sedan att spridas ungefär samtidigt som AT&T färdigställde en kommersiell version av UNIX under namnet System III.[10] Med detta uppstod den första ”dialekten” av UNIX.

Runt år 1980 började AT&T sälja UNIX som OEM-produkt[11] och därvid uppstod ett flertal nya dialekter. Bland de som licensierade källkoden till UNIX fanns det svenska företaget DIAB, Dataindustrier AB, som år 1983 började sälja sin egen UNIX-dator DS90-00 med sin egen dialekt DNIX, baserad på UNIX System V. Luxor AB å sin sida tillverkade också de en UNIX-dator med namnet ABC-1600, och en egen UNIX-dialekt med namnet ABCenix. DIAB tillverkade och sålde även en egen C-kompilator som är den enda del av verksamheten som överlevt i form av det USA-baserade företaget WindRiver Systems som idag även utvecklar ett realtidsoperativsystem, något som var en viktig del redan i den UNIX-variant som såldes av DIAB. Emedan ABC-1600 aldrig var någon stor framgång kom DS90, och dess efterföljare i DS90-nn och DIABnnnn-serierna att säljas med stor framgång till bland annat svensk stat och miltär. På detta vis kom UNIX första gången till Sverige. År 1991 köptes DIAB av företaget Bull och produktionen av DIAB-datorerna och operativsystemet DNIX lades ned.

I USA började AT&T tillsammans med Sun Microsystems vid denna tid att utveckla en mer standardiserad UNIX under namnet System V Release 4, eller SVR4 som den ofta kort och gott kallas. Denna version

[8]C kom senare att vidareutvecklas till det objektorienterade språket C++ av Bjarne Stroustrup efter idéer från programspråket SIMULA, och från C och C++ kommer den huvudsakliga inspirationen till programspråket Java.

[9]UC Berkeley, University of California, Berkeley

[10]Seventh Edition kommer före System III, dessa versionsnummer är ur olika serier och skall inte förväxlas.

[11]OEM — Original Equipment Manufacture, betecknar här en produkt som tillverkas av ett företag men säljs under ett annat företags namn.

släpptes år 1989. I början av 90-talet kom denna version att integreras med verktyg från BSD och gavs då namnet Solaris. Suns version av UNIX kallas alltjämt Solaris.

År 1993 sålde AT&T sin UNIX-verksamhet till Novell som dock snart styckade upp det hela och sålde vidare rättigheterna. Namnet UNIX förvaltas som nämnts av The Open Group (ett industrikonsortium) medan källkoden såldes till Santa Cruz Operation, SCO, som idag säljer denna "original-UNIX" under eget namn. De anställda som arbetat med UNIX på AT&T (eller rättare sagt deras dotterbolag) såldes till Hewlett-Packard.

BSD å sin sida fortsatte leva och distribueras fritt. I mitten av 1990-talet delades det dock upp i tre olika "grenar", FreeBSD, NetBSD och OpenBSD. Dessa skiljer sig åt såtillvida att FreeBSD är inriktad på maximal skalbarhet för stordrift på servrar, NetBSD inriktad på maximal portabilitet mellan olika former av hårdvara, och OpenBSD inriktad på maximal säkerhet.

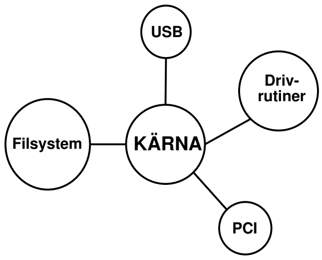
2.2 Mach, MacOS X, GNU/Hurd

I POSIX-gruppen ingår även en del andra operativsystem, undantaget de som redan nämnts. Detta avsnitt är kursivt, och är du helt ointresserad av det som nämns i rubriken kan du lika gott hoppa över det helt.

Under början av 1980-talet diskuterades inom den datorvetenskapliga disciplinen en mängd upptänkliga arkitekturer för konstruktion av kärnor för operativsystem. Precis som när det gäller arkitektur för husbyggnation finns det på detta område inte något entydigt svar på hur en bra operativsystemarkitektur ser ut. I vissa kretsar uppstod en konsensus runt att s.k. *mikrokärnor* var framtidens melodi, och att alla operativsystem hädanefter skulle byggas enligt denna princip.

Mikrokärnorna skulle hantera grundläggande hårdvaruabstraktion, minnestilldelning till program som körde i datorn, samt hantering av processer (vilket vi kommer till längre fram i boken). Kärnan skulle vara så liten, kompakt och abstrakt som möjligt, medan mer avancerade ting som filsystem skulle hanteras av externa *servrar*, program som körde för sig själva, separat från kärnan, och som kunde stängas av, startas om, krascha o.s.v. utan att påverka stabiliteten i själva kärnan. (Se illustrationen i figur 2.2.) Det finns inga direkta bevis för att detta skulle vara bra, bara en allmänt hållen argumentation för att ökad abstraktion ofta ökar stabilitet och utvecklingstakt.

Mach, *QNX* och *L4* är alla exempel på sådana mikrokärnor. Linux däremot är ingen mikrokärna, den har drivrutiner för hårdvara och filsystem inkompilerade i kärnan och kallas därför för en *monolitisk* kärna.



Figur 2.2: En mikrokärna och omgivande komponenter som inte är en del av kärnan. QNX har t.o.m. drivrutiner för hårdvara utanför kärnan, Mach har det inte. Jämför med figur 1.3.

Linux kan emellertid byta ut hela moduler i kärnan under drift på samma vis som du byter legobitar i ett legohus, det är inte så att kärnan skulle sakna manöverbarhet bara för att den är monolitisk.

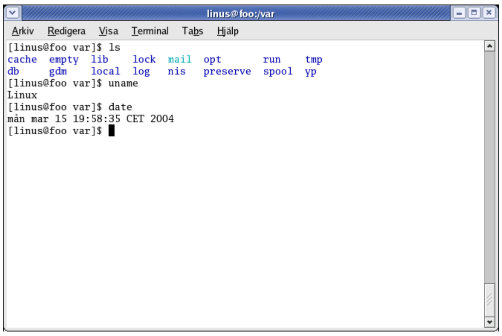
Kärnan *Mach* utvecklades under 1980-talet vid Carnegie-Mellon University i USA och förutspåddes en lysande framtid. Den föll i slummer några år och vidareutvecklades sedan något vid University of Utah till Mach 4.

Emellertid orkade få operativsystemtillverkare försöka sig på att implementera de servrar som Mach behövde för att fungera ordentligt enligt mikrokärnepricipen. Istället valde flera utvecklare att kompilera in stora delar av BSD direkt i kärnan, helt i strid med mikrokärnekonceptet, och skapade på så vis en ny UNIX-liknande kärna. Denna filosofi tillämpades i såväl IBM:s OS/2 som i NeXTSTEP, vilket i sin tur ligger till grund för Apple:s senaste kärna Darwin, vilken är identisk med den kärna som används i operativsystemet MacOS X. Även om dessa projekt ofta skryter med sin mikrokärnearkitektur är det oklart hur korrekt detta egentligen är, den som bekymrar sig om saken kan med fördel studera detta akademiska spörsmål och skapa sig en egen uppfattning i frågan.[12]

Den enda kärna som verkligen använder Mach på det sätt som var tänkt är GNU-projektets i dagsläget ofullständiga kärna HURD.[13] Tanken med detta projekt var ursprungligen att det snabbt skulle bli klart, just därför att det använde en mikrokärna som enligt detta tänkesätt skulle göra det enkelt att felsöka systemet. Denna målsättning om snabbhet har misslyckats, men HURD utvecklas fortfarande och skall enligt

[12] Se exempelvis comp.os-research FAQ [7] för utförliga diskussioner.

[13] HURD utläses ”HIRD of Unix-Replacing Daemons”, där HIRD står för ”HURD of Interfaces Representing Depth” o.s.v.



Figur 2.3: Ett vanligt skal (bash).

planen bli kärnan i GNU OS, bestående av HURD och GNU-projektets övriga program. HURD:s Unixersättning är för övrigt tänkt att vara portabel mellan olika mikrokärnor, och vissa försök har gjorts för att få den att köra ovanpå L4.

2.3 POSIX innehåll

POSIX-standarden innehåller en stor mängd definitioner. Detta rör sig främst om biblioteksfunktioner i programbibliotek för språket C som måste stödjas för att kunna uppfylla standarden. För den ordinarie användaren är detta inte speciellt spännande, och denna bok handlar om att *använda* GNU/Linux, inte om att programmera det.

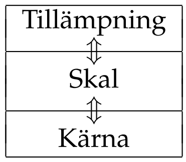
För den vanliga användaren är det ett antal specifika egenskaper som utmärker ett POSIX-system, vilka avhandlas i följande avsnitt.

2.3.1 Skal och kommandon

Bland det första du erfar i ett POSIX-system är betydelsen av ett *skal* (engelska: shell). Andra beteckningar är *kommandotolk*, *kommandorad*, eller slarvigare: *prompt*, vilket egentligen bara betecknar en rad i ett skalfönster av den typ som visas i figur 2.3. Användare tenderar att känna igen ett skal när de ser det. Skalet är en interaktiv öppning in i din dator och ger dig fullständig tillgång till all information om systemets tillstånd. Härifrån kan du starta och avsluta program, döda program som har stannat eller starta om datorn. I princip allt systemunderhåll kan skötas från ett skal. Skal kan öppnas från alla former av terminaler och över nätverk via protokollen SSH eller Telnet.

Om du har svårt att lokalisera terminalen i ditt system, så bläddra gärna fram till avsnitt 2.3.11 på sidan 105 och läs i förväg så att du kan hitta den.

Öppningen in i din dator som erbjuds av skalet ser ut något åt det här hållet:



Detta skall tolkas så, att användaren via skalet kan starta olika tillämpningsprogram, och skicka kommandon till kärnan.

Alla dessa ”öppningar” in i datorn kallas ofta slarvigt för ”terminaler”, ett ord som kommer från den tid då datorer var placerade i ett datorrum, och enkla enheter med skärm och tangentbord placerades ut på olika arbetsplatser för att möjliggöra åtkomst till själva datorn via en lång sladd. Denna terminologi har kommit att hänga kvar, och alla öppningar in i datorn, oavsett om de kommer från tangentbordet och skärmen på din PC, över nätverket eller via serieporten, kallas helt sonika för *terminaler*. I dessa terminaler kan du sedan interagera med ditt skal.

Skalen har den egenheten att de ofta används som programspråk för att skriva enkla datorprogram, skript. Detta är ett vanligt användningsområde, och känns igen från operativsystem som exempelvis CP/M eller DOS där det var möjligt att skriva så kallade ”batchfiler” (.BAT) som kördes i kommandotolken. Alla operativsystem har i princip skriptspråk av detta slag men döljer det ibland på olika listiga vis. AppleScript, VBScript, AREXX o.s.v. är andra skriptspråk som varit populära genom åren. Ibland är de skilda från skalet och ibland inte. Vi skall titta närmare på skript längre fram.

POSIX specificerar att ett skal skall finnas tillgängligt på alla system under namnet **sh** (som i **shell**) samt en rad karakteristika för detta skal. Skal kommer dock i olika smaker:

Almquist Shell eller **ash** är en klon av Bourne-Again Shell (se nedan) skriven av Kenneth Almquist, och avsett för att snabbt köra skript skrivna i bash. Det är inte avsett att användas interaktivt.

Bourne Shell eller **sh** (inga andra bokstäver, eftersom det var det första) utvecklades av Steven Bourne vid AT&T Bell Laboratories. Detta är det äldsta skalet, och från detta program kommer ordet ”shell” som används om skalprogram.

Bourne-Again Shell eller **bash** är GNU-projektets implementation och utökning av Bourne Shell. Det inkluderar idéer från såväl Korn shell och C shell (se nedan).

C Shell eller **csh** utvecklades av Bill Joy vid University of California. Det innehåller inslag av syntax från programspråket C i syfte att göra skalet mer programmerbart då det används som skriptspråk.

Korn Shell eller **ksh** är det skal som påminner mest om POSIX-specifikationen, det utvecklades av David G. Korn vid AT&T Bell Laboratories.

TENEX C Shell eller **tcsh** är en utökning av C Shell som utvecklats med operativsystemet TENEX som förebild. Det är kompatibelt med C Shell och i själva verket oftast det du får upp om du startar programmet **csh**.

Z Shell eller **zsh** kombinerar delar av Korn Shell, C Shell och TENEX C Shell med en del egna uppfinningar. Z Shell skrevs ursprungligen av Paul Falstad under hans studietid vid Princeton University. Detta är förmodligen det mest populära skalet efter Bourne-Again Shell.

Det skal du får upp om du inte specifikt frågar efter något annat brukar vara Bourne shell (t.ex. på BSD-varianterna) eller Bourne Again Shell (i GNU/Linux). Många användare av POSIX-system lägger stor vikt vid betydelsen av att använda det ena eller andra skalet, men du kommer inte ifrån att om du använder flera olika system så är Bourne shell det du alltid måste kunna hantera, om inte annat så för att kunna installera ditt eget favoritskal. (Se även avsnitt 2.3.4 för att se hur du bestämmer standardskal för din eller andras användare.)

Den som använder ett fönstersystem som t.ex. *X* behöver kanske inte använda skalet till dagligdags, men det är i dagsläget absolut nödvändigt att kunna hantera ett skal för att på allvar bemästra ett POSIX-system. I Microsoft Windows-världen är skal inte helt obekanta, men användare som kommer till POSIX-världen från Macintosh kan vara mycket ovana vid att använda skal och stöter därmed på en tröskel.

I ditt skal bör du använda *tabkomplettering*. Detta innebär att när du börjat skriva någonting som operativsystemet borde kunna gissa sig till fortsättningen på, så slår du till tabulatortangenten. Om du till exempel skriver **cat /etc/pas** och slår till tabulatortangenten ($\rightleftharpoons$), så kommer ditt skal i normalfallet att själv komplettera de sista tecknen så att kommandot **cat /etc/passwd** blir synligt på skärmen. Tabulatorn kan även gissa sig till namn på kommandon. Tabulatorknepet är direkt nödvändigt att

kunna för att snabbt och flinkt kunna manövrera i skalet och du bör lägga några minuter på att experimentera med det och nöta in det.[14] Tabbkomplettering finns i alla civiliserade skal som någonsin skapats.

En annan egenhet i skal rör piltangenterna. Uppåt- och nedåtpilen på tangentbordet (↑↓) låter dig vandra genom kommandohistorien och upprepa sekvenser som du tidigare har slagit in. Det är ofta smidigt att kunna återanvända äldre kommandosekvenser genom att bara bläddra upp ett par steg och trycka **ENTER**.[15] De flesta skal kan även upprepa gamla kommandon genom att skriva två utropstecken (!!) för att beteckna föregående kommando. Du kan exempelvis upprepa föregående kommando och lägga till något du glömde: med **!! -a** kommer föregående kommando att upprepas med växeln **-a**.

Utöver detta har de flesta skal vissa mindre egenheter, så du bör läsa dokumentationen för det skal du tänkt (eller tvingats) använda för att ha full kontroll över det.

Det vanligaste och första du gör i ett skal är att ge kommandon till operativsystemet. Kommandon i POSIX-system skrivs med kommandoväxlar av bindestreckstyp, och har formen:

kommando [-a] [-b] [-c växelargument] [-d | -e] [operand...]

Vi tolkar detta abstrakta exempel:

- **-a** och **-b** är enkla växlar ("gör a", och "b") En *växel* ändrar funktionen hos ett kommando. De flesta kommandon kan användas utan växlar, men ofta nog behövs de.

- **-c växelargument** är en växel med en parameter **växelargument**. Detta innebär, att förutom att växeln **-c** aktiveras, skickas någon form av extra information med som behövs för just denna växel.

- **[-d | -e]** betyder *antingen växel "d" eller växel "e"* — inte båda.

- **Operanden** till sist är vanligtvis en fil som operationen skall utföras på.

Klamrarna ovan **[foo]** betyder genomgående att denna del är frivillig, dvs ej nödvändig.[16]

Ett par exempel:

[14] Se denna skalövning som POSIX-motsvarigheten till en skalövning.

[15] Denna funktion fanns även i senare versioner av MS-DOS under namnet "DOSKEY", och fick aktiveras för hand.

[16] Syntaxen är besläktad med reguljära uttryck, se avsnitt 2.3.6.

```
# ls
Mina dokument       foo.txt       bar.txt
```

Kommandot **ls** (som betyder *list*) visar filer i en katalog. Inga växlar anges.

```
# ls -l
drwxr-xr-x   5 sofie   sofie   4096  1 mar 00.05 Mina dokument
-rw-rw-r--   2 sofie   sofie    128  3 aug 20.49 foo.txt
-rw-rw-r--   9 sofie   sofie    256 24 jul 23.09 bar.txt
```

Här anger den enkla växeln **-l** att jag vill ha *långt format* av samma information. Därför får jag också samma information som tidigare, plus en massa extra detaljer som jag bett om att få se.

I vissa fall kan enkla kommandon användas med flera växlar trunkerade: **kommando -abcDxyz** vilket aktiverar alla växlarna **a,b,c,D,x,y** och **z**. Här är ett exempel:

```
#ls -al
drwxrwxr-x   2 sofie   sofie   4096 23 jun 20.27 .
drwxr-xr-x  84 sofie   sofie   4096 14 aug 20.25 ..
drwxr-xr-x   5 sofie   sofie   4096  1 mar 00.05 Mina dokument
-rw-rw-r--   2 sofie   sofie    128  3 aug 20.49 foo.txt
-rw-rw-r--   9 sofie   sofie    256 24 jul 23.09 bar.txt
```

Här anges både att långt format önskas på utskriften, och att *alla* filer skall visas, vilket resulterar i att de normalt dolda katalogerna **.** och **..** syns.

Växlarna är känsliga för stora bokstäver. Själva kommandona består som regel av två till nio *små* bokstäver. Listigt nog har de mest frekvent använda kommandona givits korta namn så att de skall gå snabbt att skriva. Detta gör ibland att kommandospråket känns kryptiskt, du kanske har svårt att intuitivt gissa namnet på ett kommando utan får nöta in det. Vinsten ligger i den snabbhet du uppnår efter viss övning.[17]

Här följer några grundläggande kommandon som du måste känna till för att komma igång i ett POSIX-kommandoradssystem.[18] En del

[17]Det skall noteras att vissa av kommandona är inbyggda i skalet, så att de alltid finns tillgängliga om du har ett skal. Typiskt nog är till exempel **ls** *inte* inbyggt i skalet, medan **cd** däremot är det. Du kan således gå omkring i filstrukturen men inte se någonting, om bara kommandoskalet fungerar. De övriga kommandona (som **ls**) är (liksom skalet) vanliga datorprogram som ligger i filsystemet, oftast i katalogen */bin* eller */usr/bin*. (Se avsnitt 2.3.5 för mer information om programmens placering i filsystemet och om filsystemet i största allmänhet.)

[18]POSIX kallar inte dessa kommandon för kommandon, utan *verktyg* (engelska: *utilities*).

admin	alias	ar	asa	at	**awk**
basename	batch	bc	**bg**	break	c99
cal	**cat**	**cd**	cflow	**chgrp**	**chmod**
chown	cksum	cmp	colon	comm	command
compress	continue	**cp**	crontab	csplit	ctags
cut	cxref	**date**	**dd**	delta	**df**
diff	dirname	dot	**du**	**echo**	ed
env	eval	ex	exec	exit	expand
export	expr	false	fc	**fg**	**file**
find	fold	fort77	fuser	gencat	get
getconf	getopts	**grep**	hash	head	**iconv**
id	ipcrm	ipcs	jobs	join	**kill**
lex	link	**ln**	**locale**	**localedef**	logger
logname	lp	**ls**	m4	mailx	make
man	mesg	**mkdir**	mkfifo	**more**	**mv**
newgrp	**nice**	nl	nm	nohup	od
paste	patch	pathchk	pax	pr	printf
prs	**ps**	**pwd**	qalter	qdel	qhold
qmove	qmsg	qrerun	qrls	qselect	qsig
qstat	qsub	read	readonly	renice	return
rm	rmdel	**rmdir**	sact	sccs	**sed**
set	**sh**	shift	sleep	sort	split
strings	strip	stty	tabs	**tail**	talk
tee	test	time	times	touch	tput
tr	trap	true	tsort	tty	type
ulimit	umask	unalias	**uname**	uncompress	unexpand
unget	uniq	unlink	unset	uucp	uudecode
uuencode	uustat	uux	val	**vi**	wait
wc	what	who	write	xargs	yacc
zcat					

Figur 2.4: Verktyg (kommandon) som definieras av POSIX-standarden. De som jag av någon anledning ansett vara extra viktiga är markerade med fetstil.

begrepp i följande framställning är kanske inte alldeles uppenbara och förklaras på annan plats i boken, så tag dem för vad de är eller referera till bokens index om något är för svårt att förstå. En fullständig lista över POSIX-standardkommandon återfinns i figur 2.4. Fler kommandon kommer att presenteras efter hand som vi avhandlar de områden de berör.

Dessa kommandon har ett visst grundläggande beteende som är definierat i POSIX-standarden. Utöver detta innehåller de olika tillgängliga implementationerna av dessa ofta en rad utökningar, så för varje nytt system du använder bör du undersöka vilka speciella egenheter kommandot har på detta system. Till exempel har GNU-projektets implementationer ofta många specialparametrar och finesser som kan vara bra att känna till.

cat foo.txt slänger ut filen *foo.txt* i skalet, dvs i din terminal, ditt fönster. Om nu målfilen inte skulle vara en textfil kan resultatet bli mindre lyckat. Ordet *cat* kommer från engelskans *catenate*, som betyder "koppla efter varandra", "seriekoppla", "slå samman" vilket också är ett vanligt användningsområde för kommandot, vilket vi skall titta närmare på i avsnitt 2.3.9. Testa gärna något enkelt i stil med **cat /etc/passwd**.

1 Användarkommandon, verktyg
2 Systemanrop (för programmerare)
3 Bibliotekanrop (i programbibliotek)
4 Linux: speciella filer (såsom filer i */dev*-hierarkin av filsystemet), BSD: kärngränssnitt
5 Filformat för konfigurationsfiler, standarder, miljöer, makron
6 Spel, demonstrationer, illustrationer
7 Linux: drivrutiner och liknande för hårdvara, BSD: diverse information
8 Systemadministrationsmanualer, privilegerade kommandon (sådant som återfinns i systemets */sbin*-kataloger)
9 Kärnutvecklingsriktlinjer, referenser
n Inbyggda kommandon i Tcl/Tk

Figur 2.5: Tillgängliga manualsektioner för **man**-kommandot i Linux och BSD.

cd foo byter aktuell katalog, folder, mapp eller vad du nu önskar kalla det till *foo* **cd** utläses *change directory*. Den fördefinierade katalogen: ".." (som i **cd ..**) – går ned en nivå i trädet, d.v.s. ett steg närmare roten. Allmänt hållet exempel: **cd /etc** gör att du ställer dig i katalogen */etc*. (För mer information om dessa mystiska filsystemsting, läs avsnitt 2.3.5.)

cp foo bar utläses *copy* och kopierar en fil från *foo* till *bar*. Målet kan vara en katalog, i så fall kopieras filen dit med sitt gamla namn. Vanlig variant: **cp -R foo/ bar/** — kopierar ett helt katalogträd med underkataloger och allt.

date ger aktuellt datum och aktuell tid. Ofta vill du använda din dator som armbandsur och skriver därför **date**. Detta kommando kan dock tyvärr inte ordna romantiska träffar.

echo "foo" skriver ut (engelska: *echoes*) texten *foo* på terminalen. Ganska meningslöst kan tyckas, men detta kommando är viktigare än det först ser ut. Vi kommer att använda **echo** en hel del senare.

ls efter engelskans *list* listar filerna i aktuell katalog. Detta är ofta det första kommando du skriver, ibland behöver du visa extra fileinformation och skriver då **ls -al** vilket ger detaljerad information om filer, länkar med mera.

man sektion foo ger dig tillgång till en manualsida om kommandot, biblioteksfunktionen, demonen o.s.v. med namnet *foo*. Det frivilliga argumentet *sektion* anger vilken typ av manualsida du är ute efter, det kan nämligen finnas två eller flera manualsidor för samma uppslagsord ibland. (Strängt taget är inte sektionsoperanden en del av POSIX-standarden, men används i princip i alla POSIX-system.) En lista över olika sektioner finns i figur 2.5.

Varför dessa underliga sektioner? För att förstå detta skall du föreställa dig att dokumentationen till UNIX-system traditionellt levererades i ett antal olika tjocka ringpärmar, som var och en dokumenterade en viss del av operativsystemet.

Dessa ringpärmar svarar mot de olika sektionerna bland *man*-sidorna. **man 7 signal** kan således tolkas som ”slå upp sidan om signaler i ringpärm 7, den som handlar om drivrutiner och liknande”. Om du bara skriver **man signal** kommer detta att motsvara att kommandot letar efter signaler i ringpärmsordning, först i pärm 1 med användarkommandon o.s.v. Sökandet upphör när den första manualsidan med namnet ”signal” påträffas. (I just detta fall stannar du på ett GNU/Linux-system i sektion 2 om systemanrop, trots att det kanske var sektion 7 du var ute efter.)

Med **man -k nyckelord** går det att söka efter ett nyckelord i *alla* manualsektioner, vilket kan ta lång tid (och ger då ofta en omfattande träfflista, som sedan i sin tur får besökas med man-kommandot). Om inte *man* ger önskad information kan du prova med:

info foo ger dig tillgång till en *texinfo*-sida om kommandot (etc.) *foo*. Skillnaden mellan **man** och **info** är att medan **man** är en POSIX-standard för manualer är *texinfo*-manualerna en GNU-specifik informationstjänst, som bland annat har utökat stöd för att framställa tryckta versioner av dokumentationen. *texinfo*-informationen är ofta mer omfattande, välskriven och pedagogisk, men är inte alltid tillgänglig. De flesta POSIX-system du stöter på har med stor säkerhet en mängd GNU-program installerade, och om de är ordentligt installerade ger *texinfo* ofta den information du behöver. För mer information om GNU-projektet, se avsnitt 5.1.

mkdir foo skapar i aktuell katalog en ny katalog med namnet *foo*. Du kan sedan kopiera filer till din nya katalog eller t.ex. gå ned i den med **cd foo**.

mv foo bar flyttar filen *foo* till *bar*. Detta kommando används även för att byta namn på filer, du så att säga flyttar dem från ett filnamn till ett annat, **mv gammaltnamn nyttnamn**. Givetvis kan du också

flytta filer till andra kataloger med exempelvis **mv foo.txt /foo/bar/fnord/**. Om du vill flytta en hel katalog går det också bra, med **mv foo/ bar/** flyttas katalogen *foo* ned i katalogen *bar*.

rm foo tar bort *foo*. Vanliga varianter: **rm -rf foo** — tar bort *foo* oavsett om detta är en fil eller en katalog, och *allt* som eventuellt ligger i den katalogen, om det är en katalog. Kommandot **rm** är precis livsfarligt, det är förenat med stora risker att skjuta sig själv i foten. Du måste dock använda det en hel del.

rmdir foo samma sak fast enbart för kataloger. De flesta börjar snart att använda **rm -rf foo** istället för detta kommando.

uname -a visar information om det POSIX-system som du just nu befinner dig i. Typiskt resultat: `Linux felicia 2.4.20-9 #1 Wed Apr 2 13:42:50 EST 2003 i686 i686 i386 GNU/Linux` Detta visar att du kör *Linux* på en dator med namnet *felicia*, att Linuxkärnan har version 2.4.20 och att det är den nionde upplagan av kärnpaketet. Datumet talar om när kärnan är kompilerad. i686/i386 är namnet på olika maskinkodsuppsättningar[19] från Intel och säger att du nog kör en form av Pentium III (vilket är en typisk produkt med stöd för instruktionsuppsättningen i i686-arkitekturen). De tre ix86-siffrorna indikerar maskintyp, processor och plattform. Skillnaden mellan de tre är av intresse för få. Här är ett annat möjligt resultat:
`SunOS fnord 5.8 Generic_108529-14 i86pc i386`[20]

exit avslutar ditt skal och stänger terminalen där det fanns. Det fungerar även att hålla inne knappen **Ctrl** och trycka på **d**, om du står i början av en rad.

vi är en texteditor, se avsnitt 2.3.7. Du kommer att behöva använda någon form av texteditor ganska mycket.

2.3.2 Processer

Vi behöver nu lära oss något om hur datorprogram körs i ett POSIX-system. Ett centralt koncept för att förstå detta är begreppet *process*. Processer (engelska: *process* eller *task*, som i *multitasking*) är datorprogram som körs i en dator.

[19] Ibland kallade ISA, Instruction Set Architectures.

[20] *Solaris* (SunOS) på maskinen *fnord* version 5.8 (kallad ”Solaris 8”) för Intel-PC med i386-liknande processor.

När vi i dagligt tal säger "datorprogram" så kan vi mena olika saker: programmet utskrivet på papper, dess beståndsdelar (algoritmer), det fönster vi ser på skärmen när vi startar ett datorprogram eller den fil som innehåller programmets körbara kod, och som har ett visst filnamn som oftast är detsamma som programmets namn.

En process kan däremot inte existera utanför datorns minne, utan består av det avtryck ett program gör i operativsystemet när det körs. En process kan startas, stoppas, fortsätta där den stannade, och dödas. När den dödas försvinner programmets avtryck ur operativsystemet och minnet den använde kommer att frigöras.

Tekniskt sett består en process av ett par minnesområden och en programstack i datorns minne. Detta är av intresse för programmerare, men inte för användare. Programmerare intresserar sig också för *trådar*, även kallade lättviktsprocesser, vilket är delar av en och samma process, avsedda att exekveras parallellt. Exempelvis kan programmet **grip** läsa av ett ljudspår från CD-skiva och samtidigt konvertera ett tidigare läst spår till MP3.[21] Dessa två parallella uppgifter sker i två olika trådar. Att tillverka program som arbetar på detta vis kallas *jämlöpandeprogrammering* (engelska: *concurrency programming*). Vi skall här dock fokusera på "riktiga" processer.

POSIX-systemen är i allmänhet multiprocessystem, vilket betyder att flera processer, datorprogram, kan köras parallellt i operativsystemet, precis som en människa kan cykla och tugga tuggummi samtidigt. Grafiskt sett kan detta ofta upplevas som flera fönster där det händer saker samtidigt: en film spelas upp i ett fönster medan du ordbehandlar i ett annat fönster (varför du nu skulle vilja göra det).

Processhanteringen i POSIX-systemen är *preemptiv*, vilket innebär att vissa processer kan stannas mitt i arbetet, vänta en tid och sedan aktiveras igen. Detta är viktigt eftersom datorns processor (om den bara har en) bara kan hantera en process åt gången. Den kan då ge illusionen av att flera processer körs samtidigt genom att växla mellan dem i rask takt, vilket också är vad som sker i de flesta datorer. (Om du verkligen har flera processorer i din dator kan programmen naturligtvis verkligen köra samtidigt också.)

Tidigare versioner av Microsoft Windows[22] saknade stöd för preemptiv multiprocessering, och fick istället förlita sig på att processerna frivilligt lämnade ifrån sig processorn när andra program behövde ha tillgång till den. Detta innebar att när ett program kraschade, hängde sig, kraschade samtidigt hela operativsystemet och datorn fick startas om. Med preemptiv multiprocessering är det nästan alltid möjligt att

[21] Se vidare avsnitt 11.7.3 på sidan 398.

[22] Windows 3.11, Win16

undvika sådana situationer, och hos POSIX-systemen har detta alltid varit en naturlig ingrediens.

Processer kan starta nya processer. Om du skriver ett kommando i ett skal startas detta som en ny process, om det inte är fråga om ett inbyggt kommando. Processer har också *ägare*, vilket är identisk med en viss användare i systemet, vilket kan vara en fysisk person, root-användaren, eller en abstrakt användare av något slag (se avsnitt 2.3.4).

En process som startats av en annan process kallas för ett *barn* (engelska: *child*) till denna process, och den process som startat en annan process kallas följdaktligen för dess *förälder* (engelska: *parent*). Detta släktskap kan gå i flera led.

Att starta en process kallas att *grena* (engelska: *fork*) från föräldern, och att avsluta en process som inte vill avsluta frivilligt kallas för att *döda* (engelska: *kill*) processen. Anledningen till att det kallas för att "grena" är att exekveringsvägen delas i två delar, en för den gamla processen och en för den nya processen, som tilldelas ett nytt processnummer. Ibland används även ordet *gaffla* om processgrening, eftersom översättningen av det engelska ordet *fork* är lite otydlig.

Denna terminologi skall inte övertolkas. Det är exempelvis fullständigt i sin ordning att ett barn dödar sin förälder utan att detta får oidipala konsekvenser, och processer förökar sig uppenbarligen hermafroditiskt.

En process kan i huvudsak befinna sig i endera av två lägen: *körande* (running) eller *sovande* (sleeping). Att en process är *körande* betyder att den arbetar och utför beräkningar, uppspelning, kopiering, utskrift eller vad den processen nu är avsedd att göra. En *sovande* process väntar på att någon annan process skall bli klar med något, eller på att den på något annat vis skall anropas. Om du undersöker processlistan på ett POSIX-system kan du ofta se mängder av processer, men oftast är det bara två-tre stycken som verkligen kör, de övriga ligger och sover och tar inte kraft från datorns processor. Alla processer tar dock minne i anspråk och allt fler bonusprocesser ökar behovet av stora mängder minne.

Varje process i ett POSIX-system har ett ID-nummer, kort och gott kallat PID - *process ID*. Dessa ID-nummer kan återanvändas, men så länge processen kör eller sover kommer den att behålla sitt ID-nummer.

Från skalet kan du alltid som användare starta nya processer, d.v.s. kort och gott *köra datorprogram*. Skriver du till exempel kommandot **ls** så kommer detta att starta programmet **ls**, och det kommer att köras som en egen process. Denna process kommer dock att vara bunden till det skal där du startade processen, och all in- och utmatning kommer också att ske i detta fönster. Detta kallas för en *förgrundsprocess*.

Om du vill starta processen parallellt med ditt övriga arbete i skalet, kan du skriva ett &-tecken efter kommandot. I fallet med **ls** är detta inte så meningsfullt, men om du exempelvis startar en webbläsare vill du normalt inte att denna skall blockera allt övrigt arbete i skalet, och du skriver därför **mozilla&**. Mozilla kommer då att startas ”bredvid” skalet, och du kan fortsätta skriva saker i ditt skal. Skalet kommer då ofta att skriva ut den nya processens jobbnummer, eller barnnummer (vilken process i ordningen som startats från detta skal) inom klamrar, och sedan det globala processnumret, t.ex. **[1] 16654** eller liknande siffror.[23] En sådan process kallas för en *bakgrundsprocess*. Felutskrifter och meddelanden från det startade programmet kommer dock även i fortsättningen att hamna i skalet där det startades, så du bör inte starta program som skriver ut stora mängder text på detta vis. Ibland kan det se ut som att skalet ”låst sig”: du ser några utskrifter men ingen prompt. Om detta sker ska du prova att slå på tangenten **ENTER**ett par gånger, prompten kommer då ofta tillbaka. Om detta ändå inte fungerar kan processen dödas genom att du håller ned tangenten **Ctrl** och trycker på tangenten **c** (**Ctrl+c**).

Skalet där du startat dina processer kallas *gruppledare* och de barnprocesser, som startats i detta skal tillhör en och samma *processgrupp*. Denna terminologi gäller bara processer som har startats i någon terminal (vilket är platsen där skalet i sin tur körs). Om du dödar gruppledaren kommer hela processgruppen att dö. Detta innebär att om ditt skal avslutas, till exempel om du loggar ut, kommer alla processer som du startat i skalet också att dö. Det hela kan liknas vid ett tågset: om loket som kör längst fram stoppas av signalsystemet stannar alla vagnarna.

Processer kan styras och kontrolleras från skalet medan de kör. Följande kommandon används för att kontrollera processer:

ps listar alla processer som kör i ditt skals processgrupp. När du skriver kommandot dyker en kort lista över processer upp, något i stil med:

```
# ps
  PID TTY          TIME CMD
16612 pts/1    00:00:00 bash
16636 pts/1    00:00:11 emacs
16696 pts/1    00:00:00 ps
```

Som synes ligger gruppledaren, skalet, överst. Två processer har startats utöver denna: *emacs* har startats i bakgrunden med kom-

[23] Barnnumret, jobbnumret, har också viktiga funktioner. Vi kommer snart att komma in på hur jobbnummer används.

mandot **emacs&** och sedan har programmet **ps** körts, vilket resulterar i ännu en process. (När du ser resultatet har processen 16696 dock redan avslutats — **ps**-kommandot är ju färdigt med sitt arbete!) PID-kolumnen visar processernas ID-nummer, TTY indikerar i vilken terminal processerna kör (se avsnitt 2.3.11 för mer om detta) och TIME säger något om hur mycket CPU-tid processen har förbrukat, d.v.s. hur länge operativsystemet varit upptaget med just denna process. Om processen inte gör något beräkningsintensivt så kommer denna siffra att vara låg, som för processerna ovan. När processerna sover tar de som bekant ingen CPU-tid, och de flesta processer sover faktiskt för det mesta. emacs-processen ovan har varit vaken i sammanlagt 11 sekunder. De övriga processerna har använt så lite tid att denna tid kan avrundas till 0.

Kommandot har några standardväxlar:

ps -a listar alla processer som kör i någon *terminal* på det aktuella systemet, d.v.s. som används av någon användare.

ps -A listar alla processer som över huvud taget körs på systemet.

ps -d listar alla processer som körs på systemet med undantag för gruppledare.

ps -l visar en *detaljerad* listning av processerna.

ps -u användare listar de processer som tillhör en viss användare. (Se vidare avsnitt 2.3.4 om användare.)

Dessa växlar kan kombineras så att exempelvis **ps -Al** visar en detaljerad listning över alla processer som kör i systemet, ett kommando som en systemadministratör (root-användare[24]) ofta får tillfälle att använda.

kill PID används egentligen för att skicka signaler till en process. Själva namnet på kommandot antyder vad den grundläggande typen av signaler har för funktion i POSIX-system: de används för att på ett eller annat sätt slå ihjäl processer. Signaler kallas ibland även *asynkrona händelser*.

Det finns ofta en hel uppsjö signaler implementerade i ett visst system, och eftersom POSIX inte specificerar mer än vilka som *måste* finnas, kan du skriva **kill -l** för en obegriplig lista över tillgängliga signaler. I figur 2.6 återfinns en del signaler som alltid kan användas tillsammans med kill-kommandot.

[24]Se sidan 46

signal-nummer	**signalnamn**	**innebörd**
1	SIGHUP	”hang-up” har upptäckts på den din terminal, (en rest från den tid då folk i allmänhet använde terminaler via telefonnätet). Flera programmerare har hittat kreativa användningsområden för denna signal, ofta används den för att be demoner (se 2.3.3) att läsa om sina konfigurationsfiler.
2	SIGINT	”interrupt” — ett avbrott från tangentbordet, i praktiken betyder detta att användaren har tryckt tangentkombinationen **Ctrl+c**, vilket normalt skall avsluta programmet.
3	SIGQUIT	”quit” — ett annat avbrott från tangentbordet. Det åstadkoms med tangentkombinationen **Ctrl+**\ och är inte direkt vanligt.
6	SIGABRT	Skickas normalt *mellan* processer.
9	SIGKILL	killsignalen — om denna inte hörsammas av processen kommer processen att dödas av kärnan. Denna signal kan inte på något vis blockeras av en process.
14	SIGALRM	alarmsignal från en timer (du kan starta en process som fungerar som klocka för en annan process). Normalt skall inte en användare behöva skicka denna signal.
15	SIGTERM	Terminate, detta är den signal som normalt skickas från kill-programmet, en artig begäran till programmet om att avsluta sig självt.

Figur 2.6: Signaler som finns i POSIX-standarden, Linux har flera signaler utöver dessa.

Signalerna skickas vanligen med **kill -signalnummer PID** (där **PID** är processnumret) exempelvis det mycket berömda **kill -9 PID** som på ett relativt brutalt sätt tar död på vilken process som helst. Om du bara skriver **kill PID** så kommer signalen SIGTERM att skickas (d.v.s. detta är detsamma som att skriva **kill -15 PID**), vilket är en begäran till programmet att avsluta sig självt, och det första du skall prova då du vill stoppa en process. Om inte detta fungerar provar du i regel direkt **kill -9 PID** vilket är en begäran till kärnan om att avsluta processen med eller mot dess egen vilja.

Det är även möjligt att ange signalnamn istället för nummer, t.ex. **kill -SIGTERM PID**.

Om du vill döda programmet som kör i förgrunden i en viss terminal kan du hålla nere tangenten **Ctrl** och sedan trycka **c**. Detta kommer att skicka signalen SIGINT till alla processer som kör i förgrunden, så att de avslutas, och så att du förhoppningsvis återfår terminalprompten. Om detta inte fungerar, d.v.s. om processen "hängt sig", kan du vara tvungen att växla till en annan terminal, lista processerna för att hitta rätt processnummer (du ser ju där vilken terminal vissa processer kör på) och sedan slå ihjäl den trilskande processen med **kill -9 PID**. I själva verket kan du behöva göra detta ganska ofta om du administrerar POSIX-system och testar nya program.

I vissa operativsystem, speciellt system utformade för tidskritiska applikationer, används signaler för att skicka meddelanden mellan olika processer. I POSIX-världen var det möjligen även tänkt så från början, men det har inte blivit så. Istället har de flesta operativsystem infört ett parallellt signalsystem för detta ändamål, och de ursprungliga signalerna används bara för att signalera fel och avsluta processer.

Varför är det nödvändigt att ange processnummer, varför duger det inte med motsvarande programfils namn? Ja, om du har startat två olika instanser av ett och samma program, hur ska då systemet veta vilken av de två som skall ha signalen? Om du ändå vill döda program efter namn finns kanske kommandot **killall** för Linux, (se sidan 205). Detta kommando har dock oönskade effekter i andra POSIX-varianter: det används för att slå ihjäl *alla processer över huvud taget!*. Ett annat sådant kommando som "finns ibland" är **pkill**.

nice -n förändring program startar programmet med en förändrad prioritet. POSIX-system har normalt mer än en prioritet, men du vet oftast inte hur många. GNU/Linux har prioriteter mellan -20 och

+19, och kan alltså variera ungefär som utomhustemperaturen här i landet. Precis som människor reagerar på temperaturen så går det långsamt när det är varmt och kvickt när det är kallt. *Lågt* prioritetstal är alltså detsamma som *hög* prioritet, och vice versa.

Negativa prioriteter används bara av systemet och root-användaren, medan en vanlig användare kan skjuta prioriteterna upp och ned en smula.

Normalt startar programmen i en slags mittenprioritet (i GNU/Linux med värdet 10), så **nice -n -2 fnord** startar programmet **fnord** med lite högre prioritet och **nice -n 2 fnord** startar programmet med lite lägre prioritet.

Ditt vanliga skal kör dock i ganska hög prioritet, så normalt vill du bara skriva **nice fnord&** för att köra processen lite lagom nerprioriterad i bakgrunden, så att den inte stör andra processer.

renice -n förändring PID kan justera prioriteten på ett program under körning. Är någon process väldigt viktig kan du skicka upp dess prioritet med **renice -n -10 PID**, och är den särdeles oviktig kan du skicka ned dess prioritet med **renice -n 10 PID** så kommer den knappt att köras alls. Med **renice -g** kan en hel processgrupp omprioriteras.

En systemadministratör kan sätta prioriteter för en viss användare med exempelvis **renice -n -3 -u nobody**, om någon användare med namnet ”nobody” behöver specialbehandling.

Om du vill veta *ännu mer* om processer är detta ett bra tillfälle att bläddra fram till avsnitt 5.4.9 på sidan 202 och läsa om */proc*-filsystemet i Linuxkärnan.

Jobb

Ett annat sätt att hantera processer är att betrakta dem som *jobb*, knutna till ett visst skal. Denna kontroll är inte generell för operativsystemet men finns implementerad i de flesta skal. Internt översätter skalet jobbhanteringen till processhantering; syftet med jobben är att skapa en terminologi för att interaktivt kunna styra och påverka processer som hör till ett visst skal.

För jobbkontroll används följande kommandon:

jobs listar i allmänhet de processer som startats i det aktuella skalet. Detta ger en lista med *jobbnummer*, vilka är skilda från de process-ID-nummer, *PID* som kommandot **ps** använder, på så vis att de är

begränsade till ett visst skal, medan processnumren gäller för alla processer i operativsystemet.

```
# jobs
[1]-  Running                 emacs foo.txt &
[2]+  Running                 xcalc &
```

I denna lista ser vi att båda jobben är körande, det ena är en EMACS-session och det andra programmet *xcalc*.

Med växeln **jobs -l** kan du även få veta det vanliga, hederliga processnumret (PID), och med **jobs -p** får du *bara* processnummer, och inget annat. (Men vad är det i så fall för fel med att skriva bara **ps**?)

bg %jobbnummer skickar ett jobb till bakgrunden. Detta är ekvivalent mot att starta programmet med ett &-tecken direkt efter programnamnet. Jobbnumret som anges är inte ett processnummer (PID), utan den typ av nummer som listas av kommandot **jobs**, och som är kopplat till ett visst skal.

Om ingen parameter anges kommer jobbet överst i jobblistan att skickas till bakgrunden. Observera att jobbnumret skall anges efter ett procenttecken.

fg %jobbnummer skickar ett jobb till förgrunden. Om inget jobbnummer anges kommer det jobb som ligger överst i jobblistan att skickas till förgrunden.

disown %jobbnummer detta ”kopplar loss” ett jobb från det skal du just nu befinner dig i. Det innebär att det förlorar släktskapet med gruppledaren och i princip förvandlas till en så kallad *demon*. (För mer om demoner, se nästa avsnitt.) Ett typiskt sätt att använda detta kommando är att först sätta igång ett lågprioriterat jobb i bakgrunden med **nice foo&** där *foo* är något kommando, undersöka dess jobbnummer med **jobs**, utföra **disown** på jobbet så det kopplas loss från sessionen, och sedan logga ut. Jobbet/processen kommer då att fortsätta köra på datorn tills det blir färdigt och avslutar av sig självt, trots att den som startade jobbet har loggat ut och inte längre använder datorn interaktivt. Detta är dock lite farligt — det finns vissa jobb som av olika skäl aldrig blir färdiga, så du bör kolla att processerna inte ligger kvar och skvalpar i datorns minne flera timmar eller veckor senare.

Om du vill göra detta ofta finns det ett mera elegant sätt: kommandot **batch**. Se avsnitt 2.3.12 som börjar på sidan 107 för information om detta.

kill %jobbnummer kan användas på jobb precis som för processer. Jobbnummer skrivs ju med ett %-tecken, så **kill**-kommandot vet vad det är fråga om i vart fall. Om du vill döda alla jobb som startats i ett visst skal lite snabbt kan du göra det med **jobs -x kill**, vilket låter kommandot **jobs** gå igenom alla processerna och döda dem en i sänder.

Från en viss terminal kan du även skicka den process som just nu kör i förgrunden till bakgrunden och samtidigt *suspendera* jobbet, genom att hålla nere tangenten **Ctrl** och trycka på **z**. Detta är inte så tokigt om du vill avbryta något för ögonblicket och göra något annat. Att jobbet suspenderas innebär att det inte blir mottagligt för kommandon eller data, så ett grafiskt program som suspenderas ser helt enkelt ut att ha låst sig. I vissa POSIX-varianter finns även kommandot **stop %n** för att suspendera jobb, men detta är inte standard. Observera att suspendering inte är detsamma som att köra något i bakgrunden! Jobbet stannar helt, det fortsätter inte i bakgrunden någonstans.

Ofta kommer du att använda **Ctrl+z**-kombinationen när du glömt att starta ett program med **programnamn&**. Du får ett upptaget skal, men vill ha tillbaka kontrollen, så du trycker **Ctrl+z**. Efter att programmet suspenderat så skriver du **jobs** för att kolla vilket jobbnummer programmet har, och sedan låter du det köra vidare i bakgrunden med **bg %n** där **n** är jobbnumret. (Oftast ligger det suspenderade jobbet överst i jobblistan, och du kan skriva bara **bg**.)

Ett suspenderat jobb kan lätt återupptas om du skriver **fg** utan argument: det suspenderade jobbet hamnar nämligen först i jobbkön och kommer automatiskt att återupptas om inget argument anges till **fg**.

Om du vill avsluta själva skalets process, d.v.s. helt enkelt ta död på den process där du så att säga ”befinner dig”, kan du antingen skriva **exit** eller trycka **Ctrl+d**.

2.3.3 Demoner

Demoner (engelska: *daemon*, *demon* eller *service*) är en speciell sorts processer. Sinnebilden här är inte en diabolisk varelse med spetsig svans och horn utan *Maxwells demon*, ett tankefoster inom fysiken som tänktes kunna minska entropin i ett slutet kärl med gas genom att vid en skiljevägg i detta kärl släppa igenom enstaka molekyler på ett rådigt vis. En demon är på samma sätt en process som väntar på att något specifikt skall inträffa, och när detta sker vidta vissa specifika åtgärder.

Demoner har alltid funnits i UNIX- och POSIX-världen, och även långt före denna, i bland annat ITS (Incompatible Timesharing System) och vid forskningslaboratoriet Xerox PARC.

Demoner har hand om diverse underhållsuppgifter och tjänster. Programfilerna som demonprocesserna startas ifrån har ofta (men inte alltid) namn som slutar på bokstaven **d**, som i demon. De vanligaste demonerna och deras användningsområde illustreras bäst genom exemplifiering — notera att detta inte är några standardprogram som alltid finns, det är bara exempel:

atd är demonen som har hand om s.k. atjobb, saker systemet eller dess användare vill ha utförda vid ett visst schemalagt tillfälle.

crond är demonen som har hand om s.k. cronjobb, småsaker som systemet eller enskilda användare vill ha utförda med regelbundna intervaller. (Se vidare avsnitt 2.3.12 om såväl at- som cronjobb.)

httpd är ett generiskt namn på webbservern, oftast identiskt med programmet Apache, men inte nödvändigtvis. Demonen serverar webbsidor till de som surfar till systemet med en webbläsare av något slag. Har du inte demonen startad så kan datorn inte heller agera webbserver. (Se vidare avsnitt B.4.)

nfsd tillhandahåller filsystem över nätverket enligt protokollet NFS. (Se B.2.)

xdm, gdm, kdm är olika skärmhanterare (engelska: *display manager*) ansvariga för att starta fönstersystemet *X* och ser dessutom till att det inte stannar. Ibland heter denna process "gdm-binary" eller något annat vitsigt. (Se vidare kapitel 6.)

Strikt sett definieras ofta demoner som processer som har processen **init** som förälder.[25] (Se även avsnittet 5.4.3 om hur Linuxkärnan kan befinna sig i olika körnivåer.) Du kan lista alla demoner enligt detta kriterium på de flesta GNU/Linux-system med kommandot **ps -Al**, och leta efter processer med PPID (förälderns process-ID) satt till 1, eftersom **init** alltid har process-ID 1. I ett fullgrafiskt system kan det vara ganska många demoner som snurrar.

Förutom att demoner är responsiva, väntande program, har de också egenheten att snabbt kunna klona sig själva om det behövs. Av denna anledning väljer programmerare ofta att skapa servrar i form av demoner som grenar sig ("forkar") eftersom detta är betydligt snabbare än att starta ett nytt program helt vid sidan av. När ett program grenas

[25]Det är ur programmeringsperspektiv enkelt att skapa en sådan process: du skriver ett program som grenar av sig självt i en annan process ("forkar") och sedan avslutar sig självt ("begår självmord"). Eftersom processen då inte längre har någon förälder övergår den till barnhemmet, d.v.s. processen **init**.

kopieras nämligen bara delar av dess minne i datorn till ett annat minnesområde och startas som en ny process.[26] Detta är anledningen till att du exempelvis ofta hittar många vilande demoner med samma namn (exempelvis **httpd**). Dessa ligger och väntar på något lämligt arbete att utföra, och delar sedan på arbetet enligt någon form av ködisciplin.[27] Ofta finns det då en övre gräns för hur många sådana "slavdemoner" som skall startas.

2.3.4 Användare

Användare på ett POSIX-system är detsamma som *loginnamn*, *konton*, *profiler* eller vad du nu vill kalla det. I korthet är det en säkerhetsmekanism som gör det möjligt att skilja olika användare i systemet från varandra. POSIX-systemen är av hävd fleranvändarsystem, så som de flesta operativsystem för stordatorer var förr i världen.

För den som har vuxit upp med våra vanliga hemdatorer och persondatorer kan detta eventuellt vara en nyhet. De första hemdatorerna, från Altair 8800 över Apple II, Commodores hemdatorer, ABC-80 till den till sist helt dominerande IBM PC-arkitekturen, var i regel enanvändarsystem. En dator betydde en användare, nämligen den person som satt framför skärmen, med fingrarna på tangentbordet.

De som arbetat på ett större företag har säkerligen erfarenhet av fleranvändarsystem av något slag. Från den enklaste 3270-terminal till någon större IBM-dator, till Novell-nätverk med delade filareor som kan användas av flera personer samtidigt. Gemensamt har alla dessa system att de fordrar någon form av *inloggning*, med ett kryptiskt användarnamn och ett lösenord. I POSIX-världen har detta alltid varit ett naturligt inslag. Processer och filer har, och har alltid haft, *ägare*. Dessa ägare är detsamma som användare.

En användare har en uppsättning *privilegier* eller *rättigheter*. Dessa innefattar rätten att komma åt vissa resurser på systemet, vilket i slutänden ofta är detsamma som olika former av filrättigheter, vilket vi skall titta på längre fram. Det kan vara ett privilegium att exempelvis skriva ut papper på skrivaren, läsa och skriva till vissa delar av filsystemet, att använda ljudkortet för att spela upp ljud, att ansluta till modemet o.s.v. De flesta användare får exempelvis inte läsa andra användares filer hur som helst, utan sådant kräver privilegier. På detta vis förhindras

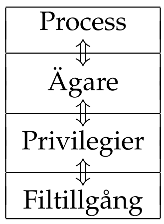
[26]Kopieringen sker med s.k. *write-invalidate*, vilket innebär att alla pekare pekare tillbaka till det gamla programmets minnesutrymmen. Så fort endera programmet väljer att skriva något till minnet kommer däremot vardera processen att tilldelas eget minne för det område som skrivits.

[27]Du kan lära dig en massa saker om köteori för servrar i Ulf Körners bok *Köteori*, se [16].

att användare avsiktligt eller oavsiktligt förstör för varandra.

Användare kan vara av två slag, *konkreta* och *abstrakta*. En konkret användare är något mycket enkelt — det är en fysisk person som behöver använda datorn, så som du eller jag. POSIX-systemen tillåter att användare använder datorn på distans, genom anslutning via någon form av datornätverk, så de konkreta användare som finns på ett system är inte alltid identiska med olika personer som kan tänkas sitta framför just denna dators skärm och tangentbord. För riktigt stora system tillhör detta rent av ovanligheterna: stora organisationer gömmer i källarvalv stora maskiner med namn som "terminalservrar" eller "applikationsservrar" som används på distans av olika användare, som i princip aldrig befinner sig på samma plats som datorn.

Abstrakta användare är mestadels användare som har med systemets interna funktioner att göra, dessa användare brukar i allmänhet äga vissa *demoner*. Anledningen är att operativsystemet vill begränsa demonprocessernas möjligheter att ändra saker i systemet. En demon är ju ett slags process, och alla processer har knutet till sig en ägare. Privilegier är i sin tur knutna till ägare. Så genom att tilldela en process en ägare, och sedan tilldela ägaren vissa privilegier, kan operativsystemet så att säga tilldela en viss demonprocess vissa speciella privilegier.



Varför används då denna komplicerade manöver med abstrakta användare?

Anledningen är, att om systemet kör igång en mängd demonprocesser utan att tilldela dem vissa specifika privilegier, kommer de att ha obegränsade privilegier. I praktiken skulle detta innebära att alla demoner köras som root-användaren, en högstatusanvändare som vi snart skall titta närmare på. Detta är farligt, eftersom root-användaren har obegränsad makt över operativsystemet. Detta kan skapa kaos, om exempelvis någon demon skulle löpa amok och börja radera filer eller förstöra minnesareor för andra processer. Genom att begränsa privilegierna på detta vis, skyddas demonerna från sig själva.

root

Som nämndes är ett POSIX-system ingen polyteistisk gudavärld där olika väsen slår varandra i huvudet och konkurrerar om makten. Systemet är i själva verket monoteistiskt, det finns en användare som i detta lilla vattenglas står över alla andra användare. Denna användare är både konkret och abstrakt, den är identisk med den person som en gång installerade operativsystemet, och identisk med den användare som startar den första processen varje gång operativsystemet startar. Från root-användaren utgår alla andra användare. root-användaren skapade i begynnelsen alla processer, och root-användaren skapade de första användarna vid installationen, och varje gång operativsystemet startas så väcks de olika demonerna i systemet av root-användaren. Root-användaren kör init-processen, processen med process-ID 1, som har makt över alla andra processer i operativsystemet.

root, som även kallas *superuser* och ibland slarvigt bara *admin*, *operator*, *administrator* o.s.v. är uppenbart modellerad på vår monoteistiska Gud. Det är ett intressant tankeexperiment att fråga sig hur ett operativsystem från exempelvis Japan eller Indien, där detta maktförhållande inte är lika självklart, skulle ha sett ut.

Du kan lätt att se om du är systemadministratör på ett POSIX-system eller inte. Om du kan logga in som root, så är du det. I annat fall är du det inte. Om du inte kan göra det, är du beroende av en person som kan logga in som root för de flesta allvarligare ändringar i systemet, såsom till exempel att installera nya program och lägga till nya användare. Om du läser denna bok, bör du ha tillgång till något system där du kan logga in som root. Har du själv installerat exempelvis GNU/Linux så är du av naturliga skäl root på minst ett system.

Beträffande root-användaren gäller det att denna normalt inte skall logga in i grafiskt läge på en värddator. Anledningen är ganska enkel: i ett grafiskt system är det lätt att komma åt något av misstag och eftersom root-användaren har rätt att göra vad som helst på systemet så kan vad som helst skadas. Grafiska program kraschar oftare och kan därför orsaka skada medan de störtar, och de grafiska programmen öppnar oftar säkerhetshål, vilket sammantaget gör att det är olämpligt att root-användaren kör grafiska gränssnitt över huvud taget. Många gör dock detta ändå.

Det normala sättet att arbeta som root-användare är att från sitt vanliga konto öppna ett terminalfönster och växla till root, exempelvis med det enkla kommandot **su -** följt av root-lösenordet. Du kan sedan konfigurera systemet och liknande ting. Alla program som startas inifrån en terminal där du är inloggad som root kommer också att ägas av root, så det är inga problem att exempelvis starta grafiska konfigureringsverk-

tyg eller multipla grafiska texteditorer. I vissa fönstersystem kommer systemet automatiskt att be efter root-lösenordet när du vill köra något administrativt verktyg, och byter därefter användare själv för just det verktyget.

Användarkonton

Vanliga användare kan också ha omfattande privilegier, även om de aldrig når upp till root. I vanliga fall skall du använda datorn som en vanlig användare, det är därför installationsprogram för POSIX-system nästan alltid erbjuder dig att skapa några vanliga användarkonton under installationen. Fördelarna med att inte köra alla program som root är uppenbara — du skyddas från dig själv. Även om alla dina program skulle krascha, eller systemet helt låsa sig, kan du i normalfallet logga in som root-användaren från någon annan terminal (i Linux kan du t.ex. normalt växla mellan olika terminaler med kombinationenen **Ctrl+Alt** följt av någon av funktionstangenterna **F1** till **F6**) eller rent av från någon annan dator, och slå ihjäl de trilskande processerna som låst sig.

Hade du kört dina program som root kunde vad som helst ha hänt.

Olika POSIX-system har olika principer för att administrera användare, d.v.s. att lägga till nya användare, ta bort gamla, och ändra privilegier. En användare har i princip sju saker som behöver kunna ändras av en administratör:

- Ett **ID-nummer** även kallat **UID**. root har nummer 0, de olika demonerna har nummer mellan 1 och säg 499 och vanliga användare brukar ha nummer från 500 och uppåt. Numren är vad systemet internt använder för att koppla saker till olika användare, själva användarnamnet slås bara upp när det behövs. Administrationsverktygen för användare ordnar normalt så att du inte behöver konfronteras med dessa nummer.

- Ett **användarnamn**. Detta skall inte innehålla svenska bokstäver eller vara särskilt långt. Mer än 8 tecken skall det normalt inte vara. I större organisationer kan detta dock bli nödvändigt i alla fall.

- Ett **fullständigt namn**. Vad användaren egentligen heter. För en konkret användare är detta uppenbart, för en abstrakt demonanvändare kan det vara samma som användarnamnet eller något annat fyndigt.

- Ett **lösenord** (engelska: *password*). Detta skall, åtminstone på ett nätverksanslutet system som kan kommas åt utifrån, vara svår-

gissat. Med ”svårgissat”, menas att det skall vara svårt för ett annat *datorprogram* att gissa det, och datorprogram kan som bekant testa med hela Svenska Akademiens ordlista, samt t.ex. vända alla ord som förekommer i denna bak och fram och prova med det. Ett lösenord skall inte kunna förekomma i en ordlista. Lösenordet skall anges vid inloggning i systemet, och abstrakta användare skall därför *inte* ha lösenord: de skall ju aldrig logga in på systemet, bara köra processer!

- En **hemkatalog**. Detta är den plats där användaren lagrar sina filer, om inte annat några ordbehandlingsdokument eller bokmärken du gjort i någon webbläsare. Se vidare avsnitt 2.3.5.
- Ett **standardskal**. Varje användare kan ha ett eget skal, smaken är delad. Alla skal som finns på systemet är möjliga val. Att flera användare nuförtiden inte ens använder skalet är ju en lustig sak i sammanhanget, de får ett skal i alla fall. En användare som inte har något skal kan inte logga in på systemet över huvud taget, abstrakta användare har till exempel inget skal.
- En **standardgrupp**. Varje användare tillhör minst en grupp, mer om detta i följande avsnitt.

Alla dessa uppgifter lagras normalt radvis i en fil som heter */etc/passwd*.[28] Du kan lätt titta på denna fil i vilket POSIX-system som helst med hjälp av kommandot **cat /etc/passwd**. Den kommer att se ut något åt det här hållet:

```
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
.....
kent:x:500:500:Kent Jönåker:/home/kent:/bin/bash
bertil:x:501:501:Bertil Håkansson:/home/bertil:/bin/bash
```

Från vänster till höger anges användarnamn, lösenord, användarens ID-nummer (UID), standardgruppnummer (GID), fullständigt namn, hemkatalog och standardskal. Observera att lösenorden har bytts ut mot *x* — här brukar finnas en krypterad variant av lösenordet. I modernare system står det dock ofta just *x*, vilket bara betyder att själva lösenordet istället lagras i en fil som heter */etc/shadow*. Detta sker av säkerhetsskäl — denna fil är mycket populär att kopiera för de som vill

[28] I stora datorsystem används ibland andra s.k. *namnuppslagningstjänster* med namn som YP, NIS, NIS+ eller LDAP. Se avsnitt B.1, sidan 418

kunna stjäla användarkonton från systemet genom att dekryptera lösenorden, och därför har konstruktörerna i vissa POSIX-varianter valt att flytta lösenorden till en egen fil, som skyddas speciellt: bara root får läsa från */etc/shadow*, medan alla användare på systemet alltid kan läsa från */etc/passwd*. Denna lite knöliga lösning beror på att många POSIX-program är beroende av att kunna läsa från */etc/passwd*, så den filen kunde inte skyddas.

/etc/shadow ger också ett antal nya säkerhetsflaggor till lösenorden: det är här möjligt att välja hur ofta lösenordet måste ändras, om kontot skall låsas helt ifall användaren inte använder det på mycket länge, o.s.v.

Användaradministration

Som vanlig användare och som root använder du något eller några av följande kommandon för att påverka din (och eventuellt andra användares) användarstatus. Detta är strikt taget inga POSIX-kommandon, med undantag för kommandot **who**, men i princip alla moderna POSIX-system implementerar dem i alla fall:

chsh betyder *change shell* och kan användas för att byta skal om du inte är nöjd med det du har.

passwd utläses *password*, lösenord, har samma namn som filen i */etc*, och används för att byta lösenord. root kan använda detta med en parameter, **passwd foo** för att byta lösenordet för användaren *foo*. Notera dock att root inte kan ”ta reda på” vilket lösenord *foo* har, det lagras inte okrypterat i POSIX-system. Svaret på frågan ”vilket lösenord har jag” är ”vet inte, men om du vill kan jag eventuellt byta ut det”. Ingen administratör kan se vad användarna har för lösenord, inte utan att aktivt försöka gissa eller knäcka det med något datorprogram. Vissa administratörer gör dock detta regelbundet — alltså försöker knäcka sina användares lösenord — av säkerhetsskäl. Det är bättre att administratören knäcker det än att någon annan gör det.

su utläses *switch user*, byt användare. Om du skriver detta kommando utan argument kommer systemet att anta att du vill byta till root-användaren. Användaren *foo* kan exempelvis förvandla sig till användaren *bar* med kommandot **su bar**. Systemet kommer då att fråga efter lösenordet för *bar*, och sedan startas *bar*:s standardskal, och är aktivt tills denne skriver **exit** eller trycker **Ctrl+d**. (Om root skriver **su bar** byts miljön direkt utan frågor, root är ju ändå allsmäktig.)

Ett bindestreck som parameter **su - bar** (eller bara **su -** för att byta till root) kommer att medföra att standardskalet startas för den angivna användaren. Om du inte anger detta kommer det kanske inte att upplevas som att du verkligen bytt till den andra användaren, istället är det mera så att du utför saker "som om" du vore den andra användaren. En annan sideffekt av **su - bar** är att du förflyttas till användaren *bar*:s hemkatalog, så om du ville utföra något på den plats där du nyss befann dig, krävs det att du först vandrar tillbaka till den platsen i filsystemet med hjälp av **cd**.

who ger en lista över vilka (konkreta) användare som är inloggade på systemet just nu. Vissa system svarar också på kortkommandot **w** med bara en enda bokstav, vilket ger samma eller liknande resultat. Detta kommando är populärt att skriva omedelbart efter påloggning till något system för att "spana in läget", d.v.s. se vilka andra personer som är inloggade. Detta kommando listar inte demoner.

Att administrera användare är för det mesta enkelt, då de flesta POSIX-system har någon form av specialverktyg för detta. Flera system har exempelvis ett program som heter **adduser** för att lägga till användare, t.ex. **adduser foo** för att skapa användaren *foo*. Programmet frågar sedan efter de andra uppgifterna som behövs för användaren. Andra system har helt grafiska lyxverktyg. I GNU/Linux-världen har i princip varje distribution sitt eget specialverktyg: **useradd userdel usermod** och **chage** är exempel på program du bör undersöka om de eventuellt finns tillgängliga.

Om inget verktyg finns att tillgå, får du helt sonika editera */etc/passwd* för hand enligt mallen (med hjälp av någon texteditor, se 2.3.7), sätta lösenordet för användaren med **passwd foo**, skapa hemkatalogen (se avsnitt 2.3.5 om filsystemet) på lämplig plats, och dessutom ofta editera gruppfilen */etc/groups* (se nästa avsnitt). Detta är inte direkt smidigt.

Grupper

Poängen med att dela in användare i grupper, är att du vill kunna tilldela privilegier till hela grupper användare.

En användare tillhör bara en grupp åt gången, och den normala användaren administrerar sitt gruppmedlemskap med följande två kommandon:

groups ger en lista över grupper som du är med i, så du vet vilka grupper du skulle tänkas kunna byta till. Den grupp som står först i listan är den du är med i just nu.

newgrp foo byter medlemskap till gruppen med namnet *foo*. Om du sedan skriver **groups** igen, ser du att denna grupp nu är först i listan.

Det är normalt inte nödvändigt att byta grupp för att komma i åtnjutande av gruppens privilegier, men under vissa omständigheter kan det vara önskvärt att filerna som skapas av dig hädanefter markeras som tillhörande en viss grupp. Detta löses lätt och smidigt genom att byta grupp med **newgrp**. Gruppmedlemskap är dock normalt ingen större huvudvärk på en hemdator, sådana behov brukar bara uppstå i större datorsystem.

Gruppinformationen lagras i filen */etc/group* som innehållet rader i ett format som påminner om */etc/passwd*:

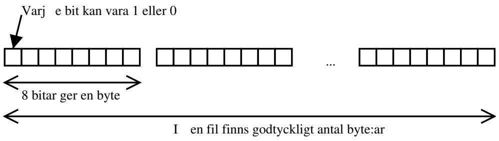
```
root:x:0:root
bin:x:1:root,bin,daemon
daemon:x:2:root,bin,daemon
.....
kent:x:500:
bertil:x:501:
```

Från vänster till höger beskrivs gruppnamn, lösenord (används sällan, tveksamt varför detta fält existerar, men en användare kan i så fall behöva ange ett lösenord för att få tillgång till en viss grupp), gruppens ID-nummer (ofta kort och gott kallat GID), och en kommaseparerad lista med namnen på gruppens medlemmar. Observera att en användares standardgrupp anges genom att skriva denna grupps ID-nummer i */etc/passwd*-filen, vilket kan verka som en underlig mix när det är själva *namnen* på gruppmedlemmarna som används i */etc/group*-filen.

Som synes har en del POSIX-system egenheten att skapa en ny grupp för varje ny användare. Anledningen till detta är att varje användare måste tillhöra minst en grupp, och eftersom de som satt samman distributionerna inte önskar skapa en grupp med exempelvis namnet *users* som alla vanliga användare skulle kunna tillhöra, har de löst problemet på detta vis istället. I större system är detta dock ovanligt, där finns en gemensam grupp för alla vanliga användare utan speciella privilegier.

2.3.5 Filsystemet

Det har sagts otaliga gånger att det övergripande kännetecknet för alla slags POSIX-system är att *allt är filer*. Detta var ett medvetet designval av UNIX' skapare Ken Thompson och har präglat dem alltsedan den första versionen av UNIX. Låt oss först klart och tydligt definiera vad vi menar med en fil, på ett näst intill matematiskt vis:



> Med en **fil** på ett datorsystem menas en ordnad sekvens av tal mellan 0 och 255. På ett datorsystem motsvaras detta av en binär talsekvens av så kallade *bytes*, som består av 8 ettor eller nollor. Filen består således av ett godtyckligt antal grupperingar av 8 stycken ettor eller nollor. Dessa sekvenser kan lagras i datorns minne eller på en magnetisk skiva, ett band, eller likvärdigt medium. En fil är något abstrakt: även om ett datorprogram upplever en fil som en ordnad sekvens betyder inte detta att den fysiska informationen i en fil är lagrad i samma ordning.

När filer lagras på en magnetisk skiva eller annat medium ordnas innehållet på denna skiva oftast i enlighet med något visst *filsystem*. Filsystemet anger hur informationen skall struktureras och lagras fysiskt. I POSIX-världen är detta tudelat: det finns ett enda system för strukturen, och flera sätt att lagra filerna fysiskt. Användaren av POSIX-system upplever systemen som likadana på alla system, men undertill kan detaljerna skilja sig. Detta skall dock inte behöva bekymra användaren.

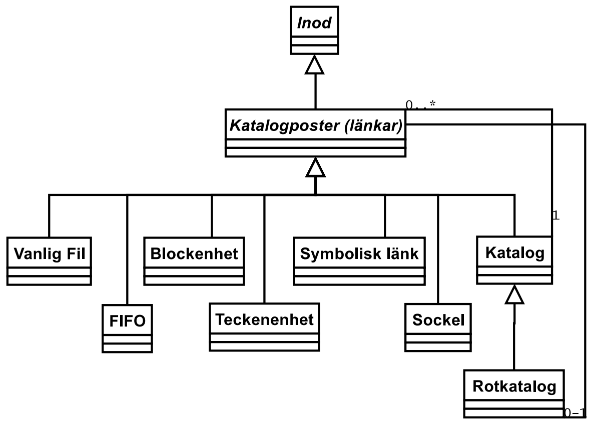
En fil kan i POSIX-världen ha i princip vilket namn som helst (och alla filer har ett namn). Det finns dock två undantag: tecknet / kan inte användas för filnamn, eftersom det används för att avdela hierarkin, och strängslutstecknet "\0" får inte heller användas i filnamn. Det senare slipper du förhoppningsvis att bekymra dig över.

En fil kan ha ett viss *filformat*, d.v.s. innehållet kan vara formaterat enligt en viss mall. Filer som inte kan läsas som text utan speciella program kallas *binärfiler*. I POSIX-världen är lyckligtvis de flesta viktiga filer som regel läsbara som text.[29]

Filsystemets hierarki

Förutom filer finns i så gott som alla operativsystem även stöd för *kataloger*. En katalog i POSIX-sammanhang är detsamma som en speciell sorts fil som innehåller katalogposter. En katalogpost kan vara en vanlig fil (av det slag vi nyss beskrivit), en katalog, eller andra speciella filer, såsom blockenheter, länkar, FIFO-köer eller symboliska länkar. Vad

[29]Eric S. Raymond beskriver detta som en av UNIX grundläggande designval: "be textual"[26].



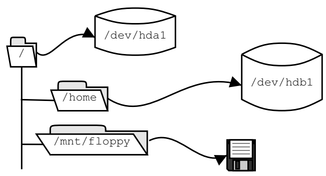
Figur 2.7: Olika filtyper i POSIX-system, *inod* (vilket inte kommer att behandlas mer i denna bok) och *katalogpost (länk)* är abstrakta ting, associationen mellan katalog och *katalogpost* (1 - 0..*) skall utläsas "en till många", en katalog har alltså noll eller flera katalogposter, och katalogposterna är någon av typerna katalog, vanlig fil, FIFO, länk, blockenhet, teckenenhet, sockel eller symbolisk länk. Rotkatalogen är speciell på så vis att den inte i sin tur finns i någon annan katalog.

dessa saker är kommer vi att gå in på senare. Viktigt att notera är att kataloger är rekursiva, d.v.s. de kan i sin tur innehålla andra kataloger. Med s.k. UML-notation får förhållandet mellan filer, kataloger och andra djur utseendet i figur 2.7.

Filsystem kan *monteras* i olika *monteringspunkter*. En monteringspunkt kan vara vilken katalog som helst, dock bara en katalog. Med montering menas att en plats på en fysisk enhet såsom en sektor på en magnetisk skiva, ett index på ett magnetiskt band o.s.v., kopplas logiskt till en katalog i filsystemet, se figur 2.8. Montering sker på olika sätt i olika POSIX-system, men ett vanligt sätt är att använda ett kommando som heter **mount**. (Du kan läsa om hur detta fungerar i Linux på sidan 213.)

Filsystemet utgår från en katalog som kallas rotkatalogen och som är den plats där operativsystemet monterar den absolut mest grundläggande katalogen. Denna katalog betecknas **/** (snedstreck) och du kan lätt gå dit med kommandot **cd /**. Passa i så fall på att undersöka vad som finns där med kommandot **ls**.

När du skriver ut sökvägar i kataloghierarkin, för att navigera dig i filsystemet, avskiljs de olika komponenterna också med ett snedstreck, som med undantag för rotkatalogen utläses "som ligger i". Exempelvis



Figur 2.8: Fysiska lagringsenheter *monteras* på kataloger i filsystemet.

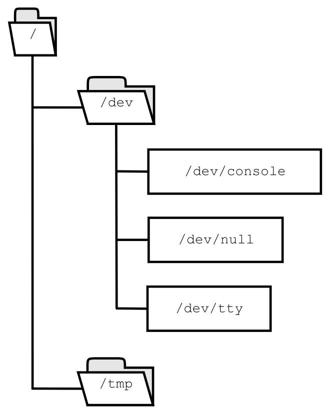
utläses sökvägen */home/bertil/foo.txt* som *"filen foo.txt, som ligger i katalogen bertil, som ligger i katalogen home, som ligger i rotkatalogen"*. I rotkatalogen finns en katalog som heter *home*, i denna finns en katalog som heter *bertil*, och i denna finns en fil som heter *foo.txt*. En sådan sökväg som inleds med /, kallas för en *fullständig sökväg* (engelska: *full path*). Sökvägar som anges relativt vart användaren befinner sig just nu, är således *relativa sökvägar* (engelska: *relative path*). Om jag exempelvis befinner mig i *home* så är den relativa sökvägen till samma fil *bertil/foo.txt*.

Att "befinna sig i" den eller den platsen i filsystemet, är att befinna sig i en viss *katalog*. *På vilket sätt* en användare befinner sig där kan variera, men ett av de vanligast sätten är att du gör det i form av ett skal, som har miljövariabeln $PWD satt till denna plats. Vi kommer att stifta bekantskap med miljövariabler senare, men kontentan är att en process (som ju skalet till exempel är) kan "befinna sig" på en plats i filsystemet genom att internt hålla reda på en sökväg, som alltid är relativt rotkatalogen. Tappar processen denna sökväg så kommer den plötsligt att befinna sig i systemets rotkatalog igen.

Platsen du "befinner dig" i filsystemet manipuleras uteslutande med kommandot **cd**. Exempelvis "förflyttar" du dig till platsen för en absolut sökväg med **cd /home/bertil/fnord** under förutsättning att *fnord* är en katalog. Du kan inte förflytta dig till en vanlig fil, eftersom det bara är möjligt "befinna sig" i en katalog.

Med untantag för rotkatalogen så finns alla objekt som kan förekomma i filsystemet — vanliga filer, kataloger, länkar o.s.v. — representerade som poster i minst en katalog. Denna struktur uppfattas ibland som ett *träd*, det talas om filträd, katalogträd o.s.v., men för POSIX-system är denna bild inte helt korrekt, på grund av att systemet kan innehålla länkar och symboliska länkar. Det korrekta datorvetenskapliga namnet för POSIX filsystem är att det är en *graf*. Om du inte vet vad detta betyder är det ingen större skada skedd.

Även om POSIX filsystem inte är något träd kan nivåerna direkt under roten ofta illustreras som ett träd. I figur 2.9 finns en trädliknande illustration av det minimala filsystem som varje POSIX-system *måste* in-



Figur 2.9: Det minimala POSIX-filsystemet.

nehålla. (Om du undersöker ditt eget system kommer du utan tvekan att finna att det innehåller betydligt fler filer.)

Dessa grundläggande kataloger och filer har följande innehåll:

/ är filsystemets rotkatalog.

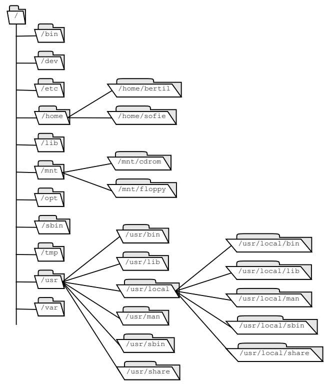
/dev är en katalog som innehåller olika blockenhetsfiler och teckenenhetsfiler. Dessa kan vara till för att kommunicera med omvärlden (exempelvis serieportar), hårddiskar, eller enheter som på annat vis behövs för systemets välbefinnande och funktion.

/dev/console är en fil som representerar systemkonsollen, vilket skall utläsas som att det är en monteringspunkt, där den fysiska skärmen och tangentbordet på datorn monteras in i filsystemet.

/dev/null är en fil som du kan kopiera strömmar av tecken till, och de försvinner magiskt! Kommandot **cat foo.txt > /dev/null** skriver ut innehållet i filen *foo.txt* på ytan av ett svart hål där ingen kan se det. Den här enhetsfilen används mest för att undvika att något skrivs ut på skärmen.[30] Om du försöker *läsa* från */dev/null*, erhålls en strid ström av filslutstecken. Operativsystemet ser denna enhet som en plats dit saker kan skickas för att aldrig återvända.

/dev/tty är en enhetsfil som representerar den första (nuvarande) terminalen. Denna går att använda som en enkel texteditor: skriv **cat**

[30] I exemplet används symbolen ”>” efter kommandot **cat**. Denna kommer att avhandlas senare i avsnittet om rör och filter på sidan 90.



Figur 2.10: Det ”inofficiella” POSIX-filsystemet.

/dev/tty > foo.txt och skriv in lite text. Avsluta med **Ctrl+D**. Titta på resultatet med **cat foo.txt > /dev/tty**.

/tmp är en katalog som kan användas för att temporärt lagra undan filer när det behövs. Alla användare (och således alla processer) kan skriva och läsa här, och skapa nya filer om det behövs.

Detta minimala filsystem existerar sällan eller aldrig i praktiken. Det ”inofficiella filsystemet” ser ut som i figur 2.10. Överallt i POSIX-specifikationen finns referenser till filer som ligger i detta filsystem: exempelvis filen */etc/passwd*. Hur skulle denna kunna existera om inte katalogen */etc* existerar först? Det går ju inte. Alltså förutsätter POSIX i själva verket ett filsystem av detta slag, och i stort sett alla system du stöter på kommer att ha denna struktur, inklusive Linux, BSD och Solaris. (Passa gärna på att i detta sammanhang bläddra fram till avsnitt 5.4.7 på sidan 199 som behandlar specifikationen för filsystemet i Linux, Filesystem Hierarchy Standard.)

Det inofficiella filträdet i figur 2.10 grundar sig på ett antal principer. Den första är *återkommande katalognamn*:

bin är namnet för kataloger som innehåller körbara *binära* filer. Programfiler kallas på engelska ibland för *binaries*.

dev är namnet för kataloger med blockenhetsfiler, efter engelskans *block devices*..

etc är namnet för kataloger som lagrar *inställningar* av olika slag, för program, skal och olika delar av operativsystemet. Varför det fått namnet **etc** efter *etcetera* finns det inget bra svar på.

lib innehåller programbibliotek, programkomponenter som kan användas av flera olika program, efter engelskans *program library*.

man innehåller manualsidor i **man**-formatet, ofta delad i underkataloger för de olika manualsektionerna och på samma vis kan kataloger med namnet **info** förekomma för GNU-dokumentation.

sbin innehåller program som används av root-användaren, som ju även kallas *superuser*, och betecknar således *superuser binaries*.

När vi vet vad dessa namn betyder då de återkommer, kan vi ge oss i kast med att tolka katalogindelningen som *inte* upprepar sig:

/ — kataloger som ligger i roten är, om de innehåller något annat än underkataloger, tänkta att bara innehålla sådant som krävs för att operativsystemet över huvud taget skall starta. */bin* är såldedes program som är nödvändiga för att starta operativsystemet, */lib* de mest nödvändiga programbiblioteken, */etc* de nödvändiga konfigurationsfilerna och */tmp* det absolut nödvändiga temporära lagringsutrymmet. */sbin* kan innehålla några program som root bara *måste* ha för att kunna hantera systemet. */dev* innehåller systemets alla block- och teckenenheter, all dess hårdvara, är en representation av själva datorn och är således också helt nödvändig.

/home är en katalog där i allmänhet 99 av 100 POSIX-system väljer att lägga användarnas hemkataloger. Denna katalog är inte sällan monterad på en speciell lagringsenhet som bara är till för att lagra användarnas data. Detta är speciellt listigt om du senare önskar byta eller uppgradera operativsystemet, då kan du ju spara den enhet som är monterad på */home*, eftersom den inte innehåller några operativsystemkomponenter.

/mnt är ett allmänt namn på ”blandade monteringspunkter”. I princip */mnt/cdrom* för CD ROM-läsaren och */mnt/floppy* för en diskettstation, men även andra enheter kan förekomma.

/opt är *optional packages* tredjepartsprogram eller extraprogram som levererats till operativsystemet. Speciellt hamnar stora, proprietära tillämpningsprogram som Oracle eller SAP R/3 på denna plats.

/usr innehåller alla delar av operativsystemet som är ”bonus”, d.v.s. de delar som inte krävs för att det över huvud taget skall starta. Här finns exempelvis ofta fönstersystemet *X* i */usr/X* eller */usr/X11R6*, en störtflod av körbara program i */usr/bin*, massor med användbara programbibliotek i */usr/lib*, fina men icke oundgängliga adminstrationsprogram i */usr/sbin* o.s.v.

/usr/local är speciellt — här lagrar du program som *inte* kommer med operativsystemet: till exempel program som du kompilerat direkt från källkod, eller som du laddat ner som binära filer och som du själv kopierat till */usr/local/bin*. När du själv kompilerar program tenderar de att på eget bevåg placera sina resulterande filer under */usr/local*. Om du aldrig gör sådana saker kommer denna katalog naturligtvis att vara tom.

/var är den plats där operativsystemet lagrar just *variabler*, vilket mestadels är saker som loggfiler (*/var/log*), där händelser i systemet eller meddelanden från olika demoner, eller elektronisk post till användarna lagras (*/var/spool/mail*).

Ovanstående är måhända lite mycket på en gång. Återkom till detta avsnitt när det passar dig och du behöver veta mer, du har möjligen redan glömt vad */usr*-katalogen är till för, men repetition är all kunskaps moder.

I alla kataloger finns det alltid två ”specialkataloger”. Den ena anges med en punkt **.** och finns alltid. Detta är en pekare till nuvarande katalog. Denna katalog kan tyckas underlig, men den används mycket, exempelvis om du vill starta program från samma katalog som du just nu befinner dig i. Du skriver då **./programnamn**. Program kan nämligen bara startas genom att ange dess absoluta eller relativa sökväg, om inte sökvägen till programmet finns i miljövariabeln **$PATH** (se avsnitt 2.3.9). Flera kommandon kräver en fullständig sökväg för att fungera, och då kan det vara lämpligt att ange **.** för att hänvisa till aktuell katalog.

En katalog med namnet **..** (punkt-punkt) finns också i alla kataloger. Denna anger närmast överliggande katalog, ”ett steg närmare roten”. Om du t.ex. står i */home/bertil* och skriver **cd ..** kommer du att stå i */home* efteråt. Dessa två specialkataloger går inte att se med kommandot **ls** men om du anger växeln **ls -a** så syns de.

Följande kommandon finns för att manipulera filsystemet, utöver **ls**, **cd** och **rm** som ju presenterats tidigare:

df utläses *disk free*, rapporterar ledigt diskutrymme för alla tillgängliga diskar. Strängt taget behöver det inte röra sig om hårddiskar: alla monterade fysiska enheters status kommer att rapporteras. En sidoeffekt av detta kommando är att alla monteringspunkter listas, vilket är minst lika användbart som att få veta hur mycket ledigt diskutrymme som finns på var och en av dem. Om du undrar vilken fysisk disk som rotkatalogen är monterad på så ger **df** svaret. (En hårddisk heter under de flesta POSIX-system något i stil med */dev/hda1*.)

find . kommer att skriva ut sökvägen till alla filer som finns under den katalog där du just nu befinner dig (.), med en rad per fil. (En absolut sökväg kan också anges som parameter.) Det kan komma ganska mycket information från **find .**, så du vill nog oftast använda det tillsammans med kommandot **grep** på följande vis: **find . | grep bar**. Detta kommer att skriva ut sökvägen till alla filer under aktuell katalog, vars namn innehåller sekvensen "bar". Vi kommer att titta närmare på denna form av konstruktioner under avsnittet om rör och filter längre fram.

Detta kommando kan användas för att utföra något som i andra sammanhang kallas för *find in files* d.v.s. sök igenom *innehållet* i ett antal filer och skriv ut vilka filer som innehåller en viss textsträng. Det uppnås på samma vis med:

```
# find . -type f -print -exec grep foo {} \; | more
```

Vilket kommer att ge en lista över alla genomsökta filer, och för varje fil kommer eventuella förekomster av den sökta strängen *foo* att listas. Om du tycker att denna metod för att söka i filer är lite omständlig är du knappast ensam, men betänk att metoden bara används från kommandoraden. Grafiska användargränssnitt har oftast egna inbyggda funktioner för att göra samma sak.

mkdir foo (av engelskans *make directory*) skapar en katalog med namnet *foo* på den plats där du befinner dig. En absolut sökväg av typen */foo/bar/fnord* kan också användas, men då måste katalogen */foo/bar* existera sedan tidigare. Bara *en* nivå i hierarkin kan skapas åt gången: */foo/bar* måste existera för att du skall kunna skriva **mkdir /foo/bar/fnord**.

rmdir foo (*remove directory*) tar bort katalogen med namnet *foo* relativt platsen du befinner dig. På samma vis som för mkdir kan en absolut sökväg anges. Sista komponenten i sökvägen anger den katalog som skall tas bort. Operativsystemet kommer oftast att protestera om du försöker ta bort en katalog som innehåller katalogposter; i så fall kan du som sagt istället använda det lite farliga kommandot **rm -rf foo**, men se upp med detta!

tar är ett kommandot som skall utläsas *tape archive*, d.v.s. "bandarkiv". Det är ett mycket användbart program som packar ihop flera filer och kataloger till ett paket och lägger det i en fil. I andra operativsystem har detta namn som "zip-filer" eller "stuffit-filer". Namnet är en kvarleva från den tid då det var vanligt att skapa säkerhetskopior av filer på ett UNIX-system genom att kopiera ut dem

till band. **tar** är ganska avancerat så vill du lära dig det ordentligt rekommenderas **man tar**. Exempel: **tar cf foo.tar bar/** packar ihop alla filerna i katalogen *bar* till ett arkiv med namnet *foo.tar*. **tar xvf foo.tar** packar upp det igen. De resulterande .tar-filerna brukar oftast även komprimeras med GNU Zip, **gzip** och får då namnet *foo.tar.gz*. Finns inte **tar** tillgängligt finns eventuellt det fossila programmet **cpio** som har en liknande funktion.

compress är det POSIX-standardkommando som finns för att komprimera filer. **compress foo** kommer att komprimera filen *foo* och ge resultatet namnet *foo.Z*. Detta kommando är dock inte speciellt vanligt, numera har i princip alla POSIX-system det något bättre programmet GNU Zip installerat.

uncompress utför omvändningen till föregående kommando.

gzip (vilket utläses *GNU Zip*) är inget POSIX-kommando men finns som sagt nästan alltid tillgängligt i alla fall. En fil komprimeras med **gzip foo** och packas upp med **gunzip foo.gz**. GNU Zip-kommandona lägger till respektive tar bort *.gz*-ändelsen automatiskt. **gzip -9 foo** gör att kompressionsgraden ökas avsevärt.

Om du har fått tag på en fil med ett namn i stil med *foo.tar.gz* kan du packa upp den helt med **gunzip foo.tar.gz** följt av **tar xf foo.tar**, eller, om du har GNU Tar installerat, med **tar xfz foo.tar.gz**. Du kan skapa en sådan fil av en katalog (och dess underkataloger) med **tar cvfz foo.tar.gz foo/**.

head foo.txt skriver ut de första 10 raderna av en filen *foo.txt*. Om du använder **cat foo.txt** kan det vara svårt att hinna se alla rader i filen, de ilar snabbt förbi på skärmen.

tail foo.txt (av engelskans *tail* = svans) visar på samma vis de 10 *sista* raderna i filen *foo.txt*.

more foo.txt är användbart när du vill inspektera en större fil en sida i taget. Sida växlas med tangenten mellanslag, och du kan bläddra ned en rad i taget med **ENTER**-tangenten. **more** är ett stort och begåvat program som du kan stifta närmare bekantskap med om så önskas (**man more**). GNU-projektet har i all spjuveraktighet kallat sin klon av detta kommando för **less** och kommandot **less foo.txt** är i själva verket vanligare än att använda **more**. (Namnet **less** är härlett från det amerikanska talesättet *less is more*.)

De tre sista kommandona fungerar bara på textfiler. Använder du dem på binärfiler kan vad som helst inträffa, men kommandona tenderar att varna för sådana misstag.

Filrättigheter

Med *filrättigheter*[31] menas vem, d.v.s. vilka användare och vilka grupper som har rätt att *läsa*, *skriva* eller *exekvera* en fil. En användare kan ha rätt att skriva men inte läsa en fil, att läsa den men inte skriva den, att exekvera den eller inte.

Att kunna *läsa* en fil innebär att kunna öppna den i en texteditor eller ett speciellt program, till exempel en ordbehandlare eller bildredigerare, kort och gott att på något vis kunna granska filens innehåll. Att kunna *skriva* en fil är att kunna lägga till saker till en fil eller helt skriva över den med nytt innehåll, eller radera den. Att kunna *exekvera* en fil (underförstått att den innehåller ett datorprogram eller skript av något slag) är att kunna instansiera datorprogrammet som lagrats i filen som en process i operativsystemet.

Filrättigheterna är den viktigaste formen för privilegiekontroll i ett POSIX-system. (Det finns andra — exempelvis får bara root-användaren lov att döda andra användares processer.) Genom att så stora delar av operativsystemet är uppbyggt av tanken att allt är filer, blir tillgången till olika resurser automatiskt en fråga om filrättigheter. Denna enkla tanke med filrättigheter för att begränsa tillgången till alla olika delar av operativsystemet har visat sig förbluffande framgångsrik och effektiv för att uppnå hög säkerhet i POSIX-system.

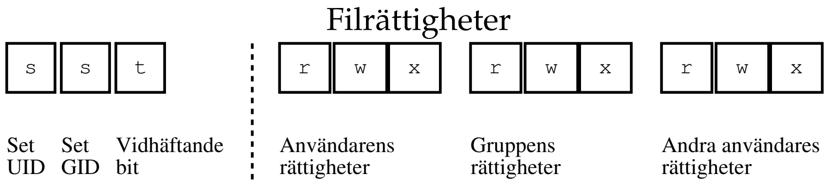
Utöver filrättigheterna har varje fil och katalog ett *ägarskap*, d.v.s. den *tillhör* någon. Filen tillhör dels en *användare* och en *grupp*. Om en användare skapar en ny fil kommer denna fil att tillhöra den grupp som användaren just då är medlem i, så som vi diskuterade tidigare.[32]

Filrättighetssystemet har uppenbarligen karaktär av stamgemenskap — systemet kan bara skilja på *egna* rättigheter, *gruppens* rättigheter och *andras* rättigheter. *Egna* rättigheter (engelska: *user rights*) avser användaren som *äger* filen, och *gruppens* rättigheter avser gruppen som *äger* filen. Den sista uppsättningen rättigheter avser rättigheter för utomstående, d.v.s. ”alla andra”.

När en användare begär tillgång av ett visst slag kontrolleras rättigheterna mot denna användare, och de grupper som denna användare är medlem i. Om användaren som begär tillgång till filen varken är ägare till den eller tillhör den grupp som äger filen, kommer rättigheterna för *andra* att tillämpas. Rättigheterna tolkas generöst, d.v.s. den störs-

[31]Med tanke på att det enda som egentligen existerar i POSIX-filsystemet är länkar borde det möjligen heta *länkrättigheter* istället.

[32]Se avsnitt 2.3.4 på sidan 44. Här ser vi i själva verket anledningen till att varje användare måste förses med en standardgrupp — om användaren skapade en ny fil och inte var medlem i *någon grupp alls*, vilken grupp skulle filen då tillfalla? Konstruktionen av POSIX är nämligen sådan att varje fil *måste* tillhöra en grupp.



Figur 2.11: Betydelsen av de olika fälten i filrättighetsfältet för en fil. **r** = läsrättighet (*read*), **w** = skrivrättighet (*write*), **x** = exekveringsrättighet (*execute*). Om du listar en katalog och ser ett **s** på den plats där **x** egentligen skulle vara (för användare eller grupp), betyder att programfilen kommer att exekveras med denna användares eller grupps rättigheter om den körs. För en katalog betyder det att alla filer som skapas i denna katalog kommer att tillhöra denna användare eller grupp. Den *vidhäftande biten* används vid sällsynta tillfällen för att skydda innehållet i kataloger.

ta rättighet som kan erhållas genom att matcha användare, grupp eller ”andra” kommer att tillämpas.

Du kan se vilka grundläggande rättigheter filer har med **ls -l**:

```
# ls -l
-rw-------  1 bertil  bertil  10249 14 jan 20.15 reglementet.pdf
drwxrwxr-x  4 bertil  users    4096 10 jun 19.27 foo
```

Filrättigheterna visas längst ut till vänster för varje fil eller katalog. De skall tolkas enligt figur 2.11, således är *reglementet.pdf* i detta fall en vanlig fil som bara får läsas och skrivas av användaren *bertil*. *foo* är däremot en katalog som får läsas och exekveras av vem som helst. (Att exekvera en katalog är detsamma som att ”gå in i den” med **cd foo**.) Bara användaren *bertil* eller medlemmar i gruppen *users* får skriva till katalogen *foo* — detta innebär i fallet med kataloger rätten att skapa nya filer i denna katalog, eller att ta bort hela katalogen.

Följande kommandon hanterar rättigheter och andra data relaterade till filer (om du plötsligt upptäcker att något i den här texten verkar övervärldsligt svårt eller obegripligt så kan du alltid trösta dig med att många professionella användare av POSIX-system *aldrig* lär sig hur rättighetssystemet fungerar innerst inne):

chown bertil foo.txt (*change owner*) byter ägaren till filen *foo.txt* till *bertil*. Av säkerhetsskäl får detta kommando bara användas av root-användaren, som på detta vis kan ta filer från en användare och ge dem till en annan. Ofta är det till exempel bra att köra **chown** på

filer som flyttas från en användares katalog till en annan användares katalog. Det är även möjligt att i ett svep byta användare och grupp för en fil med **chown användare:grupp foo.txt**.

chgrp users foo.txt (*change group*) byter ägargrupp för filen *foo.txt* till gruppen *users*, oavsett vilka som ägde filen tidigare. Detta fungerar givetvis bara om du har någon form av ägarskap till filen, samt är medlem i gruppen som du försöker byta till.

chmod <rättighet> foo.txt (av engelskans *change modes*) är ett kommando för att ändra filrättigheterna på en fil, och ett kommandot som du kommer att behöva använda tusentals gånger, så det är lika gott att lära sig allt som finns att lära sig om detta kommando direkt. Rättigheterna specificeras på flera olika vis:

chmod ug+rx foo.txt kommer att ge den ägande användaren (*user*) och gruppen (*group*) läs- och exekveringsrättigheter. **chmod o+r foo.txt** kommer att ge läsrättigheter till "andra" (*others*). **chmod a+rwx foo.txt** ger *alla* (*all*) läs-, skriv- och exekveringsrättigheter till filen. **chmod o-rwx foo.txt** tar bort alla rättigheter för "andra".

Kombinationen av **chmod**, följt av en lista utan mellanslag med **u**,**g**, **o** eller **a** direkt följt av **+** eller **-** direkt följt av en lista utan mellanslag med någon eller några av bokstäverna **r**, **w** eller **x** räcker egentligen för att sätta de flesta enstaka rättigheter som dyker upp i vardagen, med det hindrar inte att det finns andra sätt. Vill du lägga till vissa rättigheter och ta bort andra rättigheter eller bara separera rättighetstilldelningarna på ett mindre kryptiskt vis kan du skriva **chmod u+r,u+w,o-r,o-x foo.txt** för att lägga till läs- och skrivrättigheter för användaren och ta bort läs- och exekveringsrätt från "andra".

chmod a+X foo.pl (notera att detta är ett STORT **X**) lägger till exekveringsrättigheter på filen för alla användare, om *minst en* av användare, grupp eller "andra" redan har exekveringsrätt. Underligt kan tyckas, men detta är mycket användbart när du vill ändra exekveringsbarhet för många filer, exempelvis med **chmod a+rX,a-w *.pl** som kommer att ge alla användare rätt att läsa och exekvera filer som sedan tidigare är exekverbara av någon, samt slutar med filändelsen *.pl*, men ser samtidigt till att ingen kommer att ha rätt att skriva till filerna. Detta är mycket bra att använda för att ge exekveringsrätt till kataloger, som ju måste vara exekverbara för att användaren skall kunna "går in" i dem.

Det är även möjligt att tilldela rättigheter med ett =-tecken, till exempel: **chmod u=wr,go= foo.txt**, som ger läs- och skrivrättigheter till användaren som äger filen, samt tar bort alla rättigheter för gruppen och ”andra”.

Nu inställer sig en naturlig fråga: om jag tar bort skriv- och läsrättigheterna för mig själv, med **chmod a-rw foo.txt**, kan jag då inte ändra rättigheterna på filerna längre? Nej, så dumt är det inte. Du kan förvisso hindra dig själv från att läsa och skriva filer, men du kan inte ta bort rättigheterna att ändra rättigheter. En fil som du, eller en grupp som du är med i äger kan du alltid ändra rättigheterna på. Så har du lyckats låsa dig själv ute från en fil eller katalog kan du återställa rättigheterna med **chmod u+rw foo.txt**.

Rättigheter kan även anges med oktala koder, vilket inte är riktigt lika uppenbart. (Den som inte vill eller kan förstå det binära talsystemet kan hoppa över denna förklaring.) De 3x3 olika rättighetsuppsättningarna i figur 2.11 representeras i själva verket av varsin binär bit, ordnade med den mest signifikanta biten längst till vänster. Om varje rättighetsuppsättning tolkas som ett 3-bitars binärt tal kan detta skrivas oktalt som en siffra mellan 0 och 7. **r** är värd 4, **w** är värd 2, och **x** är värd 1. Kombinationerna som uppstår blir således:

Rättighet	**Kortform**	**Binärt**	**Oktalt**
inga rättigheter		000	0
exekverbar	x	001	1
skrivbar	w	010	2
skriv- och exekverbar	wx	011	3
läsbar	r	100	4
läs- och exekverbar	rx	101	5
läs- och skrivbar	rw	110	6
läs- skriv och exekverbar	rwx	111	7

Med hjälp av dessa siffergrupper kan godtycklig rättighet sättas kvickt av den som kan sina binära tal: **chmod 644 foo.txt** gör filen läsbar för alla, men skrivbar bara för användaren. En fil som skall kunna exekveras av alla men skrivas bara av den som gjort den får således **chmod 755 foo.pl**. Ett vanligt men ganska dumt trick bland nybörjare är att skriva saker som **chmod 777 *** för att slå på alla rättigheter för alla användare på alla filer. Gör inte så, för nu vet du bättre.

touch foo.txt används för att *beröra* filen. Du har kanske sett att varje fil ”minns” tiden då den sist ändrades. Med **touch** ändras den tiden

till nutid. Detta är kanske inte speciellt användbart, men **touch** har en användbar sidoeffekt: om du skriver **touch** och anger ett filnamn som inte finns, kommer filen att skapas med storleken noll! Detta används för att kontrollera en del program som letar efter en viss fil med ett visst namn för att exempelvis sluta en körning vid nästa möjliga tillfälle.

touch kan även ändra fildateringen till en fixerad tid i det förflutna eller i framtiden. **touch -t 199708290214.00 judgementday.txt** för att sätta datum och tid på filen *judgementday.txt* till den 29 augusti 1997 kl 02:14.[33]

touch har inte mycket med rättigheter att göra, men ändrar s.k. *metadata* om filen, och passar därför bra här. Om detta kommando inte berör dig kan du lämna det åt sidan.

umask <**bitmask**> (från engelskans *user permissions bitmask*) är ett kommando som sätter en s.k. bitmask, som används för nyskapade filer och oparametriserade **chmod**-kommandon. Bitmasken är ganska enkel att förstå för den som är bekant med binär aritmetik. För andra, mer normalt funtade människor riskerar denna dock att bli något av ett totalt mysterium.

Liksom med **chmod** anges bitmasken med oktala siffror, men med fyra istället för tre. (Varför skall vi snart komma till.) Bitmasken ”maskar av” behörigheter genom en logisk OCH ($\wedge$) mellan standardbehörigheter och bitmask.[34] Den standardiserade umasken som normalt används i POSIX-system brukar vara 0002, och den standardiserade rättigheten som sätts på nya filer brukar vara 666[35] vilket innebär att den standardiserade filrättigheten efter maskning kommer att blir 664, alltså läs- och skrivbar för användaren och gruppen, samt enbart läsbar för alla andra. För en nyskapad katalog är den standardiserade rättigheten 777 och efter maskning blir rättigheten på samma vis 775.

[33] Domedagen är här angiven enligt amerikansk östkusttid.

[34] Den listige läsaren tänker nu att ”men! att maska bort **w** med $2 = 010$ funkar ju inte, ty **rwx** $= 7$ och $7 \wedge 2 = 2$! Istället skulle då *bara* den biten sättas!” Men implementationen använder förmodligen inversen av masken, så den faktiska rättigheten blir $7 \wedge (2 \overline{\vee} 7) = 5$.

[35] Jag hoppas att läsaren inte är svag för talmystik. Upp 13:18

Bitmask	**Oktalt**	**Maskar bort**
000	0	inget
001	1	x
010	2	w
011	3	w och x
100	4	r
101	5	r och x
110	6	r och w
111	7	r, w och x

Om du använder kommandot **chmod** utan att ange vilka användargrupper du vill påverka, kommer rättigheterna som anges att slås på för alla grupper och därefter att maskas med umasken, så om vi antar att 0002 är aktuell umask, innebär detta att **chmod +r foo.txt** kommer att lägga till läsrättigheter till filen för alla användare, även ”andra”, medan däremot **chmod +w foo.txt** bara kommer att ge skrivrättigheter för användaren och gruppen som äger filen. **chmod +x foo.txt** kommer att ge exekveringsrättigheter till alla. På ett liknande vis används **chmod -x foo.txt** (med ett minustecken) för att ta bort exekveringsrätten för alla användare *som inte är bortmaskade*.

Vill du skrivskydda en fil till och med för dig själv finns dock en fara med umask:en: skriver du **chmod -w foo.txt** och kommer skrivrättigheterna bara att tas bort för användaren och gruppen, inte för ”andra”, på grund av umasken, så om filen tidigare var skrivbar för alla, så är det i praktiken ingen förändring! Det du i själva verket vill göra för att skrivskydda filen är att skriva **chmod a-w foo.txt**!

Såväl **chown**, **chgrp** och **chmod** tillhandahåller växeln **-R**. Denna utläses **rekursivt** (engelska: *recursive*) och är mycket användbar för att byta rättigheter på hela katalogträd: den kommer att medföra att hela katalogträdet under den punkt som angetts, såväl kataloger som enstaka filer, påverkas av ändringen.

Exempel: **chown -R anna /home/foo/** kommer att byta ägaren till *anna* på alla filer och kataloger under */home/foo/*, inklusive */home/foo/* själv. **chmod -R a+rX,go-w /usr/local/bin/** utfört av root kommer att se till att alla program installerade i */usr/local/bin* är läs- och exekverbara för alla användare, men inte möjliga att skriva över av gruppmedlemmar eller ”andra”.

En varning skall utfärdas för sådana här rekursiva anrop: om du inte är helt säker på vad du vill åstadkomma och inte har tänkt igenom det ordentligt kan en mindre katastrof inträffa. Exempelvis vill du inte

behöva återställa x-biten på alla kataloger i ett träd för hand för att du råkat ta bort den rekursivt av misstag.

Specialbitar

Utöver de nämnda rättigheterna finns även en del *speciella* rättigheter. Dessa använder sig av tre "extra" bitar som finns kopplade till filen. (Varför de kallas *bitar* inser du säkert efter övningarna med **chmod** och **umask**.) De tre extra bitarna är placerade *före* de övriga bitarna, så om du skriver rättigheterna med fyra oktala siffror istället för tre, kommer den första siffran att påverka specialbitarna, medan de tre sista fungerar likadant som tidigare.

Den första specialrättigheten är *Set-UID*-biten och *Set-GID*-biten. De här bitarna används mycket sällan, och utgör dessutom ett säkerhetsproblem, så det är tveksamt om de bör användas över huvud taget. Genom att sätta denna s.k. "s-bit" för användare respektive grupp (normalt sätter du båda) kommer en exekverbar fil (ett datorprogram) att köras som ägarens användare respektive ägande grupp, och med denna användares respektive grupps rättigheter. Om *bertil* skapar ett program som heter *foo.pl* (och som går att köra) så är filen således bertils, och om han sätter de båda **s**-bitarna med kommandot **chmod 6755 foo.pl** eller **chmod ug+rxs,o+rx foo.pl** så kommer programmet att köras som bertil, med bertils rättigheter oavsett vem som kör det. Det innebär till exempel att programmet kan läsa alla filer i bertils hemkatalog.

För kataloger har Set-GID en speciellt fiffig egenskap: de medför att kataloger och filer som skapas inne i denna katalog kommer att ärva katalogens ägargrupp, även om den användare som utför **mkdir** eller vad det nu är, är en "annan" användare, eller har växlat över till en helt annan grupp. Detta fungerar inte för att på samma sätt med Set-UID ärva ägande användare av säkerhetsskäl.

Den sista biten brukar kallas *vidhäftande bit* (engelska: *sticky bit*) eller "text-biten" och betecknas med **t**. Denna användes ursprungligen i riktigt gamla UNIX-varianter för att markera att denna fil var en textfil som användes ofta och därför skulle mellanlagras i datorns minne. Detta var långt före snabba hårddiskar och hårddisk-cache, så numera har den biten spelat ut sin roll för vanliga filer, och det rekommenderas inte att du använder den. Kataloger är inte meningsfulla att spara i minnet, och därför har **t**-biten en helt annan funktion för kataloger, och denna kan faktiskt vara riktigt nyttig:

För en katalog gäller normalt att om du har skrivrättigheter till den, så kan du dels lägga till, ta bort och ändra alla filer i den, oavsett vilka rättigheter som gäller för de individuella filerna. **t**-biten satt på en katalog betyder att även om en användare har skrivrättigheter i den kata-

logen, så får denne bara skrivrättighet på filer som denne själv är ägare till, och kan således inte ta bort andra användares filer eller katalogen själv (om denne inte är ägare till katalogen, eller medlem i gruppen som äger katalogen).

En typisk sådan katalog är */tmp*-katalogen: i denna kan alla användare skriva, men inte radera saker som inte är deras egna. Inte heller kan de radera själva */tmp*-katalogen, observera **t** i nedanstående listning:

```
# ls -dl /tmp
drwxrwxrwt  12  root  root    4096 17 jun 20.12 /tmp
```

För att sätta den vidhäftande biten för en katalog använder du kommandot **chmod 1777 foo**, **chmod a+t foo** eller liknande. Den oktala tabellen för specialbitarna blir således:

Bitmask	**Oktalt**	**Sätter**
000	0	inget
001	1	Vidhäftande bit
010	2	Set-GID
011	3	Vidhäftande bit och Set-GID
100	4	Set-UID
101	5	Vidhäftande bit och Set-UID
110	6	Set-UID och Set-GID
111	7	Vidhäftande bit, Set-UID och Set-GID

Accesskontrollistor (ACL:er)

Utöver de rättigheter som specificerades i föregående avsnitt finns i modernare POSIX-system även möjligheter till mer komplex rättighetstilldelning i form av *accesskontrollistor*. Dessa ger en mer finupplöst kontroll av rättigheter till filer.

Antag till exempel att du har två användare, *sofie* och *fabian*. Du vill att *sofie* skall kunna skriva och läsa en fil som heter *foo.txt*, *fabian* däremot bara skall kunna läsa ifrån den. Då är saken lätt — du ger *sofie* ägarskapet till filen och *fabian* sorteras in i gruppen ”andra”, som får läsrättigheter. Men antag att ingen annan än just *fabian* och ett par andra användare får läsa den här filen. Då skapar du lämpligen en grupp med namnet *fooread*, ger dem läsrättighet till filen, och tar bort läsrättigheterna för ”andra”. I praktiken skulle detta kunna se ur såhär (under förutsättning att du är root):

```
# touch foo.txt
# chown sofie foo.txt
# chgrp fooread foo.txt
# chmod u=rw,g=r,o= foo.txt
```

Antag nu, att du vill att gruppen *fnord*, som innehåller en mängd prominenta medlemmar, också måste få läsa filen. Du skulle i och för sig kunna lägga till alla medlemmar ur gruppen *fnord* i gruppen *fooread*. Då växer plötsligt mängden erfoderliga administrationsuppgifter — så fort någon läggs till eller tas bort i gruppen *fnord* fordras att du tänker på att lägga till eller ta bort denna användare även i gruppen *fooread*. Du kan skriva upp detta i en pärm eller på en post-it-lapp och tänka på det varje gång något sådant skall göras i systemet.

Detta må vara hänt. Men vad händer om du har femtio olika sådana filer, fördelade på femtio olika administratörer, och med etthundratjugo olika grupper i stil med *fnord* med korsvisa behörigheter till olika delmängder av de där femtio filerna? Eller om det är hundra? Eller femtusen? I en stor organisation är sådana scenarier tyvärr inte speciellt ovanliga.

För att åtgärda sådana problem används accesskontrollistor. Dessa är kopplade till en fil och kan bearbetas med speciella kommandon. I accesskontrollistan kan du sedan räkna upp flera olika användare och grupper, och deras respektive rättigheter. På så vis kan du ge hela gruppen *fnord* tillgång till olika filer direkt utan dessa problematiska omvägar.

Du kommer bara att behöva de speciella kommandona för accesskontrollistor om du får problem i stil med de som just beskrivits. För den ordinarie hemdatorn eller en mindre organisation räcker de "vanliga" behörigheterna i POSIX mer än väl.

Kommandona **getfacl** och **setfacl** används för att kontrollera accesskontrollistor, men detaljerna kommer inte att återges här, eftersom jag anser detta vara relativt ovanligt, och det är inte speciellt svårt att lära sig principen för dessa kommandon med manualsidorna som stöd.[36]

Länkar och symboliska länkar

I många filsystem är uppbyggnaden hierarkisk och byggd direkt på metaforen om filkabinettet. Denna metafor innebär att du tänker dig ditt filsystem som ett filkabinett av den typ som fanns i äldre kontor, med

[36] För det aktuella exemplet ges gruppen *fnord* tillgång till filen *foo.txt* med kommandot **setfacl -m g:fnord:r foo.txt**.

flera hängmappar i varje låda, och med filerna som enskilda pappersbuntar i dessa mappar. I en del operativsystem illustreras filsystemet också helt logiskt med en bild av ett filkabinett. Det är förvisso sant att själva orden *fil* och *katalog* (eller *mapp* som kataloger också kallas ibland) kommer ur denna metafor.

Denna metafor må vara sann för vissa filsystem. Den är inte sann för POSIX-systemens filsystem. Dessa är *för det mesta* ordnade på det viset, men de *kan* istället vara Kafka-liknande labyrintiska filkabinett.

För en människa är det, så vitt vi vet, omöjligt att befinna sig på två ställen samtidigt. Om någon spirituell person skulle hävda sig vara förmögen till detta kallar vi det ett transpersonellt parapsykologiskt fenomen, och betraktar det med skepsis. Denna verklighetsuppfattning beror naturligtvis på att vi betraktar vår värld som materiell, och en uppsättning atomer kan inte befinna sig på mer än en plats åt gången, sådana är naturens lagar. I filernas värld är detta däremot enkelt, okomplicerat och inget speciellt märkvärdigt.

En *länk* är inget annat än en fil som befinner sig på två ställen i filsystemet samtidigt. Till exempel kan */home/sofie/foo.txt* och */home/bertil/bar.txt* vara en och samma fil. De ser ut som två olika filer, de heter olika saker, och ligger på skilda platser, men om du försöker läsa från dem, läser du samma sak på båda platserna, och skriver du till dem skrivs samma sak till "båda" filerna. En del saker fungerar inte som för andra filer: tar du bort den ena, så finns den andra kvar, och ändrar du rättigheterna, ägare eller annan metadata[37] på den ena så ändras de för den andra också.

För att skapa en länk används kommandot **ln källa mål**. *källa* är den fil som redan finns och *mål* den nya länk som du vill skapa. Denna typ av länk kallas också *hård länk*. Varför kallas den hård? Det beror på att det även finns s.k. *symboliska länkar*. Dessa skapas med kommandot **ln -s källa mål** och skall användas när det inte passar med en hård länk.

Symboliska länkar används för att referera andra filer med namn, vilket syns då du listar en katalog:

```
# ls -l
lrwxrwxrwx  1 sofie  sofie   7 23 jun 23.17 bar.txt -> foo.txt
-rw-------  2 root   root   24 23 jun 23.00 foo.txt
```

Som synes är de symboliska länkarna tillgängliga för vem som helst, men det skall bara tolkas som att vem som helst kan *följa* länkarna —

[37]Metadata är en samlingsbeteckning på olika former av tilläggsinformation till en viss informationsmängd, som inte i sig är en del av informationsmängden. Exempel: etiketten på en pärm, Hcg-beteckningen på en bok i biblioteket ägaren till en fil o.s.v.

försöker de utföra något praktiskt på en symbolisk länk kommer rättigheterna på målfilen att användas.

En symbolisk länk är speciellt bra om du ibland behöver byta ut filen som länken pekar på. Om du tar bort en symbolisk länk (i detta fall **rm bar.txt**) kommer detta inte att påverka den länkade filen.

I POSIX finns inget sagt om att hårda länkar skulle vara begränsade på något vis, men i praktiken kräver de flesta operativsystem att filer som är hårda länkar befinner sig på samma enhet, och med samma enhet menas i detta fall att de är monterade på samma monteringspunkt. Det lysande undantaget från denna regel är monteringspunkterna själva, som kan hårdlänkas både här och var, vilket är bra att tänka på i den situation att du plötsligt inser att du skulle vilja montera ett filsystem på två olika ställen.

Nu en filosofisk utflykt. Allt i ett POSIX-filsystem är i själva verket hårda länkar. En hård länk till en fil och en fil *i sig*, går inte att skilja åt. Om du gör en hård länk till en katalog kommer denna inte att kunna skiljas från en katalog. Vi anar en bakomliggande struktur: det som vi kallar vanliga filer, kataloger o.s.v. är inget mer än hårda länkar. Det enda som är lite ovanligt är när det finns länkar till filerna på mer än ett ställe. Meditera gärna en stund över detta.[38]

FIFO-köer

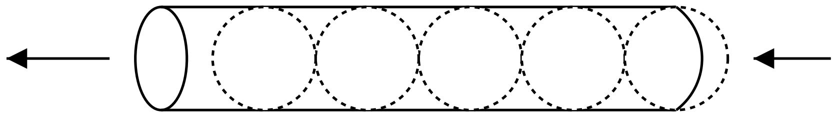
Det finns ytterligare ett slags exotiska filer: *FIFO-köer*. Namnet kommer från engelskans *First-In-First-Out* och brukar i viss litteratur även kallas för *named pipes*. Dessa fungerar ungefär som tennisbollar i ett stuprör: det du stoppar in först i ena änden trillar ut först i den andra. (Se figur 2.12) Att skriva till en sådan fil är inget speciellt, men att läsa från den är speciellt: det du läst ”trillar ut” ur filen och sväljs av programmet som läser.

FIFO-köer används normalt inte av en användare, de används för kommunikation mellan olika program, exempelvis för att skicka meddelanden till demoner. Du kan åstadkomma en FIFO med kommandot **mkfifo foo.fifo** men min gissning är att du måste vara programmerare för att komma på ett bra användningsområde för den.

FIFO-köer har mycket stora likheter med ett s.k. *rör* (engelska: *pipe*[39]), något som vi kommer att diskutera längre fram.

[38] Och när detta är klart, kan du söka vidare genom att försöka förstå vad en *inod* (engelska: inode) är för något. Att förstå det ligger utanför området för denna bok. Du kan se alla dina filers motsvarande inoder med **ls -i**

[39] Ceci, n’est pas un pipe.



Figur 2.12: FIFO-köer är som ärtrör: det du stoppar in i dem ramlar ut i andra änden.

Block- och teckenenheter

Blockenhetsfiler (engelska: *block device file*) och teckenenhetsfiler (engelska: *character device file*) är en typ av filer som i princip bara existerar i katalogen */dev* i filsystemet. Ett samlingsnamn för denna typ av filer är *enhetsfiler*. Dessa representerar saker som skärm, tangentbord, ljudutgång, grafikkortsyta, hårddiskens lagringsutrymme, disketternas lagringsutrymme o.s.v.

Blockenhetsfilerna är avsedda att ha en buffert som kan ta emot hela block av tecken, d.v.s. stora mängder bytes, åt gången, medan teckenenhetsfilerna kan ta enstaka tecken (bytes). I normalfallet är detta inget en vanlig användare skall behöva bekymra sig över.

Enhetsfilerna är öppningar in i saker som hanteras av kärnan. Som du vet hanterar kärnan alla former av hårdvara: mus, tangentbord, nätverk, serieportar och modem, skrivarportar och skrivare o.s.v. Alla dessa ting finns normalt representerade i */dev*. Det finns ingen regel om att blockenheter *måste* ligga i */dev*, det är bara en konvention.

Du kan kommunicera direkt med enklare blockenheter, som exempelvis din egen terminal, med **echo "foo" > /dev/tty**. Innebörden av just detta exempel kommer att belysas längre fram.

Enhetsfilerna skall inte tolkas som att det är möjligt att "gå in i hårddisken" genom att skriva **cd /dev/hdb** eller liknande: alla sådana manövrar kommer att sluta i besvikelse. Hårddiskar är komplicerade saker och kan inte kommas åt med mindre än att du monterar dem i filsystemet, något som åstadkoms med kommandosekvenser i stil med **mount /dev/hdb /mnt/foo**. POSIX standardiserar inte hur montering skall ske, så detta varierar mellan olika system.

För POSIX-system är hårddiskar inget annat än en lång, ordnad serie tal mellan 0 och 255, bytes. De är således exakt detsamma som en stor fil. Om det exempelvis finns en blockenhetsfil som heter */dev/hda* så representerar detta hårddiskens totala lagringsutrymme. Detta utrymme kan sedan delas i partitioner, som sedan kan formateras enligt ett visst mönster och därefter monteras in i filsystemsträdet. Alla enheter som är monterade i trädet är i själva verket för det mesta någon form

av blockenheter. (Exakt hur detta fungerar i just Linux kommer vi att gå igenom längre fram.)

Som normal användare kommer du mycket sällan i kontakt med blockenheter, annat än när det är dags att montera på någon diskenhet i filträdet. Kanske använder du någon gång någon specialenhet som */dev/null* eller */dev/zero* om du skriver skript. Det finns en enhet som heter */dev/tty* som symboliserar din terminal, och det finns ofta flera andra enheter som heter */dev/tty** (börjar på *tty*) och som symboliserar såväl terminaler som serieportar på datorn.

Det kan vara värt att notera att tillgången till saker som representeras som blockenheter följer samma regler som andra filer: du kan exempelvis reglera tillgången till datorns ljud och CD ROM-enhet bara genom att ändra filbehörigheterna till motsvarande blockenheter![40]

Socklar

Om blockenheter är ovanligt, så är socklar (engelska: *socket*) ännu ovanligare för vanliga användare. Dessa symboliserar nätverksförbindelser till en dator, och genom att stoppa in eller plocka ut tecken i en sockel uppstår en speciell sorts FIFO-kö som har den egenheten att ena änden av kön befinner sig i din dator, och den andra änden av kön kan befinna sig i en annan dator.

Socklar är oftast inkopplade från och till det egna systemet men */mnt/socketa* på din dator kan exempelvis vara samma fil som */mnt/socketb* på en annan dator. Det du stoppar in på din sida kommer ut i den andra datorn, och det den andra datorn stoppar in kommer ut på din sida.

Det finns inga direkta användarkommandon för att skapa och stänga socklar, och de finns inte kvar i filsystemet när nätverket till datorn stängs av (se vidare avsnitt 2.3.10 om nätverket) så som vanlig användare använder du dem bara indirekt genom olika tillämpningsprogram.

Den typ av socklar som går till den egna datorn är oftast av en typ som kallas *Unixdomänsocklar* (det korrekta namnet är POSIX Local IPC Sockets), medan socklar som ansluts till andra maskiner är av TCP/IP-typ. Om du vill veta hur socklar definieras i TCP/IP (vilket är den vanligaste nätverkstypen för POSIX-system) kan du bläddra fram och läsa på sidan 304.

Dolda filer och kataloger

Dolda filer i POSIX-system är detsamma som filer som har ett filnamn som inleds med en punkt, "punktfiler". Detta är bara en konvention:

[40] I Linux finns ljudenheten t.ex. i */dev/dsp*, o.s.v.

alla program och kommandon är konstruerade så att de inte visar filer som har namn som inleds med en punkt, det är ingenting som faktiskt skiljer en dold fil från en vanlig fil utöver detta. För att lista alla dolda filer i en katalog kan du ge växeln **-a** till kommandot **ls**, detta stänger av filtret som döljer filnamn som inleds med en punkt.

Även kataloger kan döljas genom att ge dem namn som inleds med en punkt.

Dolda filer och kataloger används ofta i användarnas hemkataloger för att spara inställningar för olika program. Om du använder Mozilla sparas exempelvis alla inställningar och bokmärken för detta program i en katalog med namnet **.mozilla** i din hemkatalog. Prova gärna med **ls -a** i din egen hemkatalog om du aldrig gjort det förut.

Notera att när du använder s.k. *globbning* (se 2.3.6) måste punkten inkluderas explicit för att de dolda filerna skall matchas. **ls -al [.a]foorc** listar filen ”afoorc” men inte ”.foorc” även om den finns, **ls -al .[fk]oorc** listar däremot både ”.foorc” och ”.koorc”.

Hemkatalogen ~

Hemkatalogen som är knuten till varje användare har av bekvämlighetsskäl getts kortnamnet ~: ”flygande orm”, ”tilde”, ”växelströmstecken” eller vad du nu önskar kalla det. Detta åstadkoms på ett normalt tangentbord genom att hålla nere **AltGr**, trycka på knappen som innehåller **ˆ**, **¨**, ~ och sedan på mellanslagstangenten. Beroende på system kan det resulterande tecknet se lite olika ut.

Om du vill kopiera en fil någonstans ifrån till din hemkatalog löser exempelvis kommandosekvensen **cp foo.txt** ~ detta, och vill du titta på en fil i din hemkatalog så kan **more ~/foo.txt** kanske vara en användbar kortform. Vill du kopiera en fil från *en annan användare* kan du också använda ~: **cp ~sofie/foo.txt .** till exempel.

Hemkatalogen skall följa en användare och skall normalt bara innehålla arbetsmaterial såsom några kataloger med ordbehandlingsfiler, lite bilder från digitalkameran, MP3-filer eller vad användaren nu önskar spara där. Program och andra ting som har med systemet självt att göra skall normalt *inte* ligga där. Tanken är att användaren skall kunna använda en och samma hemkatalog från olika operativsystem och olika datorer. Därför skall bara saker som är neutrala med avseende på operativsystem lagras där.

Inställningar för olika program är neutrala. Programmens inställningsfiler skall (om programmen är välgjorda!) kunna användas av samma program oavsett POSIX-operativsystem, under förutsättning att programmet finns till det operativsystemet. Exempelvis lagrar Mozilla sina bokmärken och liknande i en dold katalog i din hemkatalog, *~/.mozilla*

som sedan kan användas av alla versioner av Mozilla för alla operativsystem. Du skall alltså inte behöva en ny hemkatalog för att fixa en sådan enkel sak.

Eftersom alla arbetsfiler och alla inställningar till olika program för en användare lagras i dennes hemkatalog, är det den enda del av operativsystemet som en "vanlig" användare behöver befatta sig med. Om du administrerar ett större system med flera datorer, så se till att användarna vet hur de skall hantera sina hemkataloger för att lagra sina arbetsfiler där.

En annan finurlig sak med hemkatalogen är att när användaren vill göra en säkerhetskopia av sina filer, är det bara att kopiera hela hemkatalogen. Detta görs exempelvis med kommandosekvensen **tar cvf /tmp/ min_backup.tar ~/** . Den resulterande *min_backup.tar*-filen innehåller en komplett säkerhetskopia av alla dina filer. (Det avslutande snedstrecket är mycket viktigt, utan detta kommer alla dolda filer att tas bort från säkerhetskopian.) Ett alternativt sätt att utföra samma operation är: **cd ; tar cvf /tmp/min_backup.tar .** — du anger katalogen **.** som arkiveringspunkt, och det första **cd**-kommandot utan parameter tar dig automatiskt till hemkatalogen. (Notera att det är möjligt att utföra två eller flera kommandon efter varandra genom att sätta semikolon (;) mellan dem.)

En viss hygien rekommenderas i hemkatalogen. Ha inte onödigt många filer här, helst inga alls utan bara kataloger som i sin tur innehåller dina arbetsfiler. Utan ordning och reda i hemkatalogen kan det bli ganska jobbigt för dig att hitta det du söker.

2.3.6 Reguljära uttryck

Reguljära uttryck (engelska: *regular expressions*, *regexp*, *regex*), är även kända som "jokrar" (engelska: *wildcards*). Att använda reguljära uttryck i ett skal kallas ibland *globbning* (engelska: *globbing*), och när de används till detta har de ett lite annorlunda utseende. Det kan allmänt sägas att den som lärt sig använda reguljära uttryck i sitt arbete aldrig slutar använda dem och aldrig glömmer dem.

Reguljära uttryck är inte unika för POSIX-systemens användargränssnitt, utan används inom språkvetenskap och kompilatorteknik, specifikt inom teorin för formella språk.

Ordet "reguljärt uttryck" kommer från Noam Chomskys språkhierarki, tredje (lägsta) nivån. Denna nivå definierar språk med reguljär grammatik. Det karakteristiska för dessa språk är att de kan realiseras som en finit tillståndsmaskin, vilket i princip betyder "är lätt att programmera". Chomskys genombrott inom språkvetenskapen på

1950-talet kom att inspirera de matematiker som jobbade med formella språk, d.v.s. i praktiken programspråk för datorer.

Utifrån Chomsky formulerade John Backus och Peter Naur sin Backus-Naur Form (BNF) för beskrivning av kontextfri grammatik i formella språk, som sedan utökades av Niklaus Wirth under arbetet med programspråket Pascal, till den Extended Backus-Naur Form (EBNF) som mest påminner om reguljära uttryck så som de används i POSIX-system.

Till UNIX-världen kom reguljära uttryck då Ken Thompson lade in stöd för dem i sin texteditor **qed**, samt senare i verktyget **grep**.

Reguljära uttryck kan i olika programspråk, texteditorer och skal ha en egen dialekt och ett eget utseende. Vi skall här fokusera på den definition som gäller för POSIX, vilken till största delen kan antas gälla även för de flesta skal, kommandon och så vidare. Dessa återfinns i figur 2.13, sidan 77.

Med dessa uttryck kan du ha mycket roligt, men den främsta användningen är i texteditorer och vissa kommandoradsverktyg där du behöver ange mönster. Grundtanken är här att om du vill matcha en sträng[41] för att kunna välja ut eller manipulera delar av den. I programmet **grep** kan du till exempel skriva:

```
grep '^{}M[eo]d' foo.txt
```

Detta kommer att få till resultat att alla rader av filen "foo.txt" som börjar med ordet "Med" eller "Mod" (med stor bokstav först) kommer att skrivas ut på standardenheten. Observera att även rader som börjar med "Medan" eller "Modig" kommer att visas. Om du tar bort "^"-tecknet i början av det reguljära uttrycket, kommer även rader där strängen inte ligger först på raden att skrivas ut.

När du vill använda reguljära uttryck i ett skal skriver du till exempel:

```
ls -al *.html
```

Du menar då att du vill se alla filer med filnamnsändelsen ".html". Punkten representerar i detta fall inte "vilken bokstav som helst" utan just en punkt. Med tanke på hur mycket punkter det används i filnamn skulle det bli ganska jobbigt om alla punkter tolkades som reguljära uttryck. I skal har därför reguljära uttryck en något annorlunda syntax, som visas tillsammans med andra uttryck i figur 2.14. Att använda denna form av reguljära uttryck kallas för *globbning*.

[41] En *sträng* är en följd av tecken, en ordnad mängd. Tecknen tas från operativsystemets teckenuppsättning. "aaaaa" är till exempel en sträng bestående av fem stycken "a".

Grundläggande reguljära uttryck

Uttryck	**Betydelse**	**Exempel**
<tecken>	vilket tecken som helst matchar sig självt en gång	*fnord* matchar teckenföljden *f,n,o,r,d*.
.	matchar på *ett* tecken, vilket som helst	*fn.rd* matchar såväl *fnord* som *fnard*.
[ ]	matchar *ett* tecken som finns innanför hakparenteserna	*fn[oau]rd* matchar *fnord*, *fnard* och *fnurd* men inte *fnerd*.
[ˆ]	matchar *ett* tecken som *inte* finns innanför hakparenteserna ("allt utom detta")	*fn[ˆoa]rd* matchar alla möjliga ord som *fn.rd* matchar, men inte *fnord* och *fnard*
ˆ	matchar början av en rad	*ˆfnord* matchar ordet *fnord* men bara när det står i början av en rad.
$	matchar slutet av en rad	*ˆfnord$* matchar alla *fnord* som står ensamma på en rad, utan något annat vare sig till höger eller vänster (OBS! det får inte ens finnas mellanslag till höger eller vänster!)
\(\)	gruppera allt innanför parentesen som ett block	\(*fnord*\) matchar ordet *fnord* och grupperar det. Detta används bland annat då du vill bearbeta blocket i efterhand, till exempel byta ut just detta ord mot ett annat.
$\backslash\{x, y\backslash\}$	matcha föregående block minst x och maximalt y gånger	*fno\{3,5\}rd* matchar *fnooord*, *fnoooord* och *fnooooord*
*	matchar föregående block noll eller flera gånger	*fno*rd* matchar *fnrd*, *fnord*, *fnoord*, *fnooord* o.s.v., \(*fnord*\)* matchar *fnord*, *fnordfnord* o.s.v.

Utökade reguljära uttryck

Uttryck	**Betydelse**	**Exempel**
+	matcha föregående block *minst en* gång	*fno+rd* matchar *fnord*, *fnoord*, *fnooord* o.s.v.
?	matcha föregående block noll eller *en* gång	*fno?rd* matchar *fnrd* eller *fnord*.
( )	grupperar allt innanför parentesen i ett block	Samma som för \(\) i vanliga reguljära uttryck.
$\{x, y\}$	matcha föregående block minst x och maximalt y gånger	Samma som för $\backslash\{x, y\backslash\}$ i vanliga reguljära uttryck.
|	matcha alternativa block	*([Kk]ent | [Bb]engt)* matchar *Kent*, *kent*, *Bengt* och *bengt*.

Figur 2.13: Reguljära uttryck som är gemensamma för de flesta POSIX-system, program, skal och texteditorer. De utökade uttrycken måste normalt aktiveras genom att exempelvis ange växeln -E till ett program eller genom att använda ett speciellt program, **egrep** istället för **grep** till exempel. Observera att medan tecknen (,), { och } kan användas fritt i vanliga reguljära uttryck krävs att du använder en s.k. escape-sekvens för dem i den utökade varianten, \{ matchar { o.s.v.

Globbningsuttryck

Uttryck	Betydelse	Exempel
<tecken>	matchar ett tecken som inte är ett globbningstecken	**ls foo.txt** listar just filen *foo.txt* och ingen annan - observera att punkten inte har någon speciell betydelse här.
*	matchar vadsomhelst noll eller flera gånger, inklusive den tomma strängen (noll tecken)	**ls *.html** listar alla filer med filnamnsändelsen *.html*
?	matchar *ett* tecken, vilket som helst	**ls ???.txt** listar *foo.txt* och *bar.txt* men inte *foobar.txt*.
[ ]	matchar *ett* tecken som finns innanför hakparenteserna	**ls [bg]r[aå].txt** listar *bra.txt*, *grå.txt*, *brå.txt* och *gra.txt* men inte *krå.txt*.

Figur 2.14: Globbningsuttrycken är besläktade med reguljära uttryck men har en lite annorlunda syntax som passar skalet bättre.

Du kan ha roligt även med dessa uttryck: exempelvis listar kommandot **ls [A-Z]*[12345].?** alla filer som börjar med en stor bokstav i det engelska alfabetet, fortsätter med vilka bokstäver som helst, hur många som helst, följt av någon av siffrorna 1-5, en punkt och en filnamnsändelse bestående av *en* bokstav, till exempel *.c* eller *.h*.

Vissa användare är inte nöjda med dessa reguljära uttryck utan vill ha ännu mer möjligheter att matcha och välja ut strängar. Om sådana behov skulle uppstå finns programspråket **perl** att tillgå — detta har kraftigt utökade reguljära uttryck med alla möjliga extra tillägg.

De reguljära uttrycken kommer till stor nytta i nästa avsnitt, där vi skall se hur du kan använda rör och filter i POSIX-system för att hantera komplexa informationsmängder och hur du till sist även kan skriva enkla datorprogram, s.k. *skript* i skalet.

2.3.7 Texteditorer

> May I indent your code?
>
> — Hackerförolämpning

Förr eller senare kommer en användare av ett POSIX-system att behöva redigera innehållet i en textfil. Det finns hjälpprogram för det mesta, men när det gäller exempelvis inställningar för operativsystemet och för olika program tenderar POSIX-systemen att hålla sig strikt till textfiler, som kan redigeras av användaren. Inte minst gäller detta de så kallade ”punktfiler” (dolda inställningsfiler) som sparas i varje använ-



Figur 2.15: En terminalskrivmaskin av typen Teletype 48. Notera det s.k. traktorpappret i ovankanten där rad för rad av inmatad text dyker upp.

dares hemkatalog, och de flesta konfigurationsfiler som i ett typiskt system ligger i katalogen */etc*.

Den stora betydelsen en texteditor har för en professionell användare yttrar sig i en del egendomligheter som för en nykomling kan te sig något obegripliga.

Exempelvis är texteditorerna mestadels textbaserade program som körs, eller kan köras, helt och hållet i ett terminalfönster, och ofta styrs texteditorerna med en underlig uppsättning tangentbordskommandon. Dessa faktorer är delvis historiska: när editorerna uppfanns fanns inga grafiska fönstersystem, utan editorerna fick köras i terminalfönster.

De här visuella editorerna var på sin glans dagar (1970-talet) revolutionära: det har inte alltid varit möjligt att se flera rader av en fil på terminalen samtidigt — raderna på skärmen användes istället som buffert för tidigare kommandon, och filerna fick editeras en rad i taget. De editorer vi kommer att bekanta oss med här är dessa så kallade *visuella* editorer, där flera omgärdande rader i den fil du vill editera syns samtidigt på skärmen.

Detta att redigera flera rader samtidigt var en gång i tiden faktiskt en nyhet och betraktades med skepsis av de användare som var vana vid saker som terminalskrivmaskiner (engelska: *hardcopy terminal*, *teletypewriter*, *teletype*, se figur 2.15), där kommandon till datorn skrevs in en rad i taget precis som på en skrivmaskin, och för säkerhets skull också samtidigt skrevs ut rad för rad på ett randigt s.k. pyjamaspapper, där även utskrifterna från datorn hamnade. Kommandon och resultat

kunde sedan jämföras i efterhand.[42] I POSIX-system finns faktiskt två sådana editorer: **ed** och **ex**. Dessa får anses vara ett historiskt arv och relativt meningslösa att lära sig, och kommer därför inte att behandlas här; den som är nyfiken på ed eller ex rekommenderas att lära sig dessa via manualsidorna.

Så långt historiska anekdoter. Det saknas naturligtvis inte moderna texteditorer som är helt grafiska och fönsterbaserade.[43] Det lustiga är att erfarna POSIX-användare sällan eller aldrig använder dem. Anledningarna är flera, men huvudsakligen beror det på att styrningen med tangentbordskommandon, när användaren väl har lärt sig dem, är mycket snabbare än att manövrera menyer med en mus. Någon skulle väl hävda att dessa användare är konservativa stötar som ser upp till gamla gubbar med skägg. Så vitt jag vet är det dock hastigheten som är det övervägande skälet till att de flesta professionella användare håller sig till dessa texteditorer.

Den texteditor som ingår i POSIX-standarden kallas VI och kommer att avhandlas i ett speciellt avsnitt. Utöver denna är EMACS en minst lika populär editor. Det var uppe till diskussion huruvida EMACS skulle vara en del av POSIX-standarden, och programmets upphovsman Richard Stallman tillfrågades. Han hade då ganska dålig erfarenhet av fristående implementationer av EMACS, och bad kommittén att ta bort EMACS ur POSIX, för att slippa se dåliga implementationer: han resonerade som så att de som önskade använda EMACS ändå skulle installera hans version: GNU EMACS. Detta förklarar i någon mån varför EMACS, även om det inte är en del av POSIX, oftast betraktas som en standard och per automatik medföljer de flesta POSIX-system, dock inte alla.

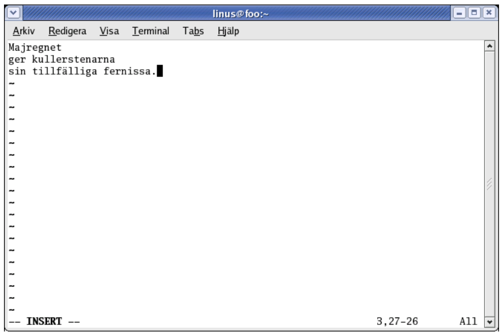
Flera professionella användare har en närmast religiös inställning till sin texteditor och missar sällan ett tillfälle att på skoj racka ned på ”den andra” editorn. Det är i själva verket inte möjligt att på objektiva grunder hävda att den ena editorn skulle vara bättre än den andra, det handlar mycket om tycke och smak. Det bästa är att behärska *både* VI och EMACS, men att kanske hålla sig till att bara lära sig en av editorerna på djupet.[44] Du måste här välja själv.

---

[42] Dessa editorer har historiskt sett sitt ursprung i teleprintrarna, de första rent textbaserade telegraferna. (De kallades också ”fjärrskrivmaskiner”.) När de första datorerna tillverkades tog konstruktörerna de delar som fanns till hands för att lösa jobbet, och teleprintern var en sådan lämplig byggsten.

[43] Fönstersystemen GNOME och KDE har exempelvis varsin egen texteditor, och ett stående skämt när en ny programmerare vill ha tips om ett bra projekt att starta för att lära sig är ”skriv en texteditor!”

[44] Det är för övrigt ett återkommande fenomen i POSIX-världen att de populäraste verktygen kommer i två dominerande par: VI och EMACS, perl och python, GNOME och KDE, GTK+ och Qt, sed och awk. Alla dessa verktyg kan, oavsett vad apologeterna säger,



Figur 2.16: En fil som editeras med VI. Editorn befinner sig i inskjutningsläge, vilket syns längst ner till vänster.

När du hoppar in i en texteditor kan det vara värt att tänka på att det alltid är möjligt att tillfälligt ta sig tillbaka ut i skalet med **Ctrl+z** och sedan återuppta editeringen med kommandot **fg** så som beskrevs i avsnitt 2.3.2 om processer.

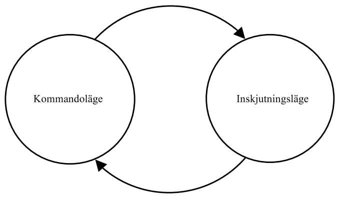
VI

VI eller **vi** utläses *visual editor* och är som sagt POSIX-standarden för texteditering. Den ursprungliga versionen av VI skrevs en gång i tiden av Bill Joy vid AT&T Bell Labs, baserat på enradseditorn **ed**. Du kan lätt börja editera en befintlig fil i VI med kommandot **vi foo.txt**. (Om filen inte redan finns kommer den att skapas.)

VI är varken intuitivt eller lätt att lära sig. Det måste omedelbart sägas. Mycket få helt textbaserade datorprogram är intuitiva över huvud taget, den som kan sin historia med exempelvis DOS-baserade ordbehandlingsprogram vet att det ofta levererades plastremsor med programmen som klistrades ovanför funktionstangenterna på datorns tangentbord för att användaren skulle kunna hitta funktionerna genom att titta där istället för i manualen. Flera användare av texteditorer för POSIX-system använder referenskort eller kaffemuggar(!) med kommandolistor för att slippa hålla alla kommandon i huvudet.

Det första du måste lära dig om VI är att den arbetar i två olika *lägen*, vilka indikeras av texten längst ner till vänster i terminalen: står det INSERT befinner du dig just i inskjutningsläge, annars i kommandoläge:

i princip utföra samma arbete som de andra.



kommandoläge (engelska: *command mode*) — detta läge används för att flytta sig runt i dokumentet och utföra kommandon som påverkar stora delar av texten. Det är alltid möjligt att ta sig till kommandoläget genom att trycka tillräckligt många gånger på tangenten **Esc** (escape) högst upp till vänster på tangentbordet.[45]

Kommandon i kommandoläget kan skrivas direkt oavsett vart markören befinner sig. Detta kan kännas ovant och farligt, men det är bara att exempelvis inleda ett kommando med att trycka ned **:** någonstans mitt inne i texten. Följande är basala kunskaper i VI:s kommandoläge:

- Du flyttar dig runt i filen med piltangenterna. Skulle detta inte fungera kan tangenterna **h j k l** användas för att manövrera markören vänster, nedåt, upp och höger.
- Starta editeringen av en fil med **vi foo.txt**. Om du bara skriver **vi** utan att ange en fil kan du läsa in en fil med **:e foo.txt**.[46] Detta fungerar också bra om du editerat en fil och vill byta till att editera en annan fil. (Spara först, annars måste du skriva **:e! foo.txt** för att ignorera de ändringar du gjort.)
- Skriv ut filen till disk (skriv över den gamla) med **:w**.
- För att avsluta editeringen av en fil: tryck först **Esc** tills du hamnar i kommandoläge och skriv sedan kommandot **:q!**. Om du har sparat innan räcker det med **:q**.
- För att ta bort innehållet i den rad du står på, tryck **D** (delete). För att ta bort raden helt och hållet, inklusive innehållet, tryck **dd** (två små **d** efter varandra). Om du gjorde något dumt och ångrar dig, tryck **u** (undo).
- För att foga in innehållet från en annan fil på den plats i bufferten där du befinner dig, skriv **:r foo.txt**. Du kan även göra exotiska saker som **:r !uname -a** för att läsa in namnet på

[45] Om nu inte **Esc** skulle finnas på ditt tangentbord kan **Ctrl+[** fungera under vissa omständigheter.

[46] Alla kommandon som börjar med **:** är ursprungligen från den gamla, radbaserade **ex**-editorn.

ditt POSIX-system (eller någon annan utmatning från vilket kommando som helst) i bufferten.

- Sök framåt efter text med **/foo** följt av **ENTER**, upprepa sökningen efter samma ord med **//** följt av **ENTER**. (Detta är samma sak som gäller i kommandot **more** eller när du läser manualsidor med **man**.)
- Sök bakåt efter text med **?foo**, upprepa sökningen med **??**.
- Sök och ersätt en sekvens med exakt samma syntax som kommandot **sed**[47] använder efter ett inledande **:** t.ex. **:s/foo/bar/g** för att byta alla förekomster av ordet *foo* mot ordet *bar*. Alternativt kan du skriva **g/foobar/s/foo/fnord/g** för att först hitta alla förekomster av *foobar* och i dessa byta *foo* mot *fnord* så att det istället står *fnordbar*.

För att gå över i inskjutningsläge, kan du trycka olika tangenter: att trycka **i** (insert) är det mest uppenbara sättet — detta går direkt över i inskjutningsläge och skjuter in tecken före den bokstav markören står över, **a** (append) gör samma sak men efter den bokstav markören står över. **I** respektive **A** gör samma sak fast i början av raden eller i slutet av raden som du just står på. **o** skapar en ny tomrad under markören, placerar markören på denna rad och går över i inskjutningsläge. Observera att det som så ofta i POSIX-världen är lite känsligt med skillnaden mellan stora och små bokstäver.

inskjutningsläge (engelska: *insertion mode*) — detta läge används för att manuellt editera innehållet i filen, d.v.s. för att skriva och redigera text för hand, utan hjälp av andra kommandon. Normalt är det fritt fram att bara skriva den text du vill ha, inklusive korrigeringar genom tryck på raderingstangenterna (delete, backspace) och liknande. Vissa implementationer av VI kommer att radera texten i bufferten samtidigt som du trycker på en raderingstangent, medan andra inte visar resultatet av sådana operationer förrän du återvänder till kommandoläge.

Du lämnar inskjutningsläget på olika sätt: tangenten **Esc** är det mest uppenbara, men i vissa varianter av VI medför alla tryck på piltangenterna också att du omedelbart lämnar inskjutningsläget.

Dessa två lägen växlar du sedan fram och tillbaka mellan under tiden du editerar din fil.

[47] **sed** förklaras ingående på sidan 95, precis som i **sed** kan VI använda reguljära uttryck.

Rudimentära kommandon	
vi foo.txt	Starta filen *foo.txt* i VI.
:q!	Avsluta VI ovillkorligen.
:wq eller :x	Skriv till fil och avsluta.
Esc	Lämna inskjutningsläge och återgå till kommandoläge. Bra att trycka på när allt annat känns hopplöst.
i	Gå över i inskjutningsläge *efter* markören.
a	Gå över i inskjutningsläge *före* markören.
I och A	Gå in i inskjutningsläge i början eller slutet av raden.
/foo	Sök *framåt* efter texten *foo*.
?foo	Sök *bakåt* efter texten *foo*.
Filhantering	
:e foo.txt	Editera filen *foo.txt* istället. Om filen som för tillfället editeras inte sparats kommer kommandot att vägra.
:e! foo.txt	Samma sak men ignorerar faran med att det du sist jobbade med kommer att gå förlorat.
:w	Skriver ut filen till filsystemet. (Sparar.)
:r foo.txt	Läser in innehållet i filen *foo.txt* på den plats där markören befinner sig.
:r !cmd	Utför kommandot *cmd*. Resultatet från kommandot (d.v.s. *stdout*) hamnar där markören befinner sig.

Sök och ersätt	
:s/foo/bar/g	Sök efter *foo* och ersätt med *bar* i hela filen utan att fråga om saker skall ersättas.
:s/foo/bar/c	Samma sak, men fråga först, innan ersättning görs.
Markering, radering och kopiering av text	
dd	Ta bort raden under markören
d3d	Ta bort tre rader under markören
yy	Kopiera raden under markören
y7y	Kopiera raden under markören och de följande 6 raderna (6+1=7)
p	Klistra in de kopierade raderna under markören
P	Klistra in de kopierade raderna före markören
	Inklistringskommandona kommer även att klistra in text som tagits bort vid radering med ”d”-kombinationer.

Figur 2.17: De vanligaste kommandona i VI. De flesta kommandona används i kommandoläget, vissa angivna i inskjutningsläget.

När du öppnat din fil syns det översta av filens innehåll på skärmen. Om filen skulle vara kortare än skärmen fyller VI utrymmet under själva filen med ett $\sim$-tecken på varje ”icke-existerande” rad.

Sedan är det bara att editera, spara, editera igen, o.s.v. En lista över användbara VI-kommandon återfinns i figur 2.17.

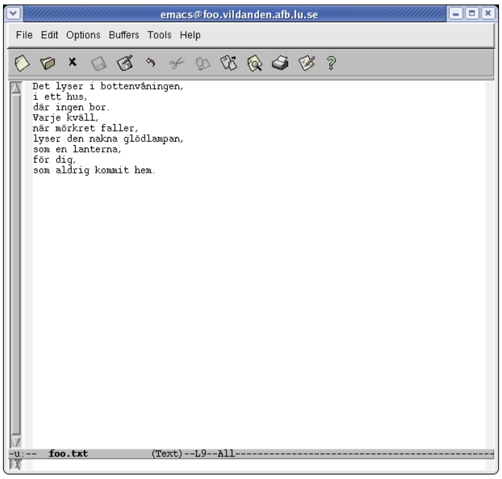
Utöver den standardiserade VI-editorn kommer många operativsystem med någon eller några extra editorer som är kompatibla med VI, exempelvis VIM - *VI Improved*. Dessa har olika egenheter men fungerar i grunden likadant som VI.

EMACS

EMACS vilket skall utläsas *Editor MACroS*, är till programstorleken, både beträffande faktisk storlek i bytes, använt minnesutrymme och komplexitet en betydlig större editor än VI, på gott och ont. Det är inte utan anledning som EMACS ibland kallas ”operativsystemet listigt förklätt till texteditor”. EMACS skrevs ursprungligen av Richard Stallman år 1976, för operativsystemet ITS (Incompatible Timesharing System) vid MIT i Boston.

Det är inte säkert att EMACS är installerat i ditt operativsystem. I så fall får du först fundera ut hur detta går till för just ditt system. Detta är en anledning så god som någon att även behärska VI.

Du kan starta editeringen av en fil med EMACS genom att skriva **emacs foo.txt**. I fönstersystem startas EMACS ofta i ett eget fönster, och det kan därför vara en god idé att skriva **emacs foo.txt&** istället, så att



Figur 2.18: En fil som editeras med EMACS. Det som visas är den version av EMACS som anpassats för GNOME.

du kan fortsätta använda skalet till något annat efteråt. Vill du prompt att EMACS *inte* skall starta i ett eget fönster skriver du istället **emacs -nw foo.txt** ("no window").

Eftersom EMACS gärna startar i ett eget fönster är det enklare att komma igång med EMACS som nybörjare. Den finns menyer för att göra det mesta som i VI måste göras med tangentkombinationer, vilket för nykomlingen kanske kan vara skönt att slippa. Det finns alltså möjligheten att snabbt komma igång med EMACS för att sedan lära sig tangentbordsmanövreringen lite efterhand. Första gången du startar EMACS (utan att ange någon fil) erbjuds du dessutom ofta en självstudiekurs i hur editorn fungerar.

När nu EMACS är ett så pass stort program föredrar många att bara ha en EMACS-session aktiv åt gången, och att därifrån växla över till de olika filer (buffertar) de vill editera. Detta är något av en smaksak i dagsläget — med dagens mått mätt är EMACS inget stort program och kan gott startas varje gång något skall editeras.

Utmärkande för EMACS är att programmet använder tangenterna **Ctrl** och **Meta** flitigt, av historiska anledningar. Tangenten **Meta** fanns framför allt på LISP-maskiner, en sorts datorer anpassade för programspråket LISP som tillverkades mestadels under slutet av 1970-talet. På våra moderna datorer har oftast tangenten **Alt** tagit över denna roll, och i vissa fall används tangenten **Esc**. I EMACS-världen kallas alla dessa

substitut för **Meta** ändå kort och gott **Meta** eller bara **M**. På datorer från Sun Microsystems motsvaras tangenten av "diamanttangenten" ⋄. I allmänhet kan du lugnt anta att **Meta** = **Alt**.

När tangentkombinationer i EMACS refereras står det ofta saker som **C-s** eller **M-x**. Detta betyder **Ctrl+s** respektive **Meta+x**, d.v.s. du skall hålla inne tangenten **Ctrl** eller **Meta** och trycka på **s** respektive **x**. Det finns ett undantag: de operativsystem som använder **Esc** som **Meta**-tangent, kräver att du först trycker *och släpper* **Esc**, därefter trycker du på önskad **Meta**-kombination.

När du startat EMACS kan du avsluta det med tangentkombinationen **Ctrl+x Ctrl+c**, vilket du utför genom att hålla nere **Ctrl**-tangenten och sedan trycka på **x**-tangenten, släppa den och sedan trycka på **c**-tangenten utan att släppa upp **Ctrl** emellan. Detta är ett nödvändigt handlag som du lär dig efterhand.

Något som är viktigt för att förstå EMACS är konceptet *buffert*. EMACS kan hantera ett godtyckligt antal buffertar för att utföra arbete i flera filer samtidigt, och har därutöver några extra buffertar. När du startar EMACS (se figur 2.18) syns två buffertar: en övre som presenterar editeringsytan för den fil du editerar, alternativt den s.k. *scratch*-bufferten, om du inte valt att editera någon fil. Nertill i EMACS-fönstret finns den en rad höga s.k. *minibufferten* som används för att skriva kommandon till EMACS.

EMACS erbjuder sedan en rad möjligheter för att se flera buffertar samtidigt och kunna växla mellan buffertar för att du ska kunna arbeta så smidigt som möjligt. EMACS skiljer sig från andra grafiska program på det viset att filer som sparats efter att de editerats färdigt fortfarande ligger och skvalpar i en egen buffert. Vill du hitta någon motsvarighet till andra programs "stäng" ("close file") är detta närmast kommandot "kill buffer" (**Ctrl+x Ctrl+k**) i EMACS.

För att EMACS skall fungera riktigt snabbt måste du lära dig snabbkommandona, men menyerna räcker som sagt långt till en början. De vanligaste tangentkommandona i EMACS finns sammanfattade i figur 2.19. Saker du förmodligen kommer att lära dig snabbt är att trycka **Ctrl+k Ctrl+k** (hålla nere **Ctrl** och trycka på **k** två gånger i rask följd) för att radera en rad och **Ctrl+s** för att söka efter text. Om du en dag önskar att något speciellt editeringskommando fanns till EMACS så visar det sig förmodligen att det redan finns om du bara orkar leta reda på det. Att trycka **Meta+x** och sedan något som du misstänker att ditt kommando kan heta, följt av ett slag på tangenten **TAB** kan ibland snabbt avslöja den mest hisnande flora av editeringskommandon.

Två ytterligare kortkommandon är viktiga att kunna: **Ctrl+_** (understreck) är kombinationen för att *ångra* senaste operationen. Håller

	Rudimentära kommandon
emacs foo.txt	Starta filen *foo.txt* i EMACS.
Ctrl+x Ctrl+c	Avsluta EMACS
Ctrl+g	Avbryt aktuellt kommando i minibufferten.
Ctrl+s	Sök efter text. Direkt efter att du tryckt Ctrl+s skall du börja skriva texten du vill söka efter, och EMACS hoppar då till första förekomsten. Avbryt kommandot med piltangenter, **Esc** eller **Ctrl+g**.
Meta+x ...	Välj godtyckligt kommando genom att skriva in det.
	Filhantering
Ctrl+x Ctrl+f	Öppna en fil i en ny buffert.
Ctrl+x Ctrl+s	Spara filen i aktuell buffert till filsystemet.
Ctrl+x Ctrl+w	Spara filen i aktuell buffert under nytt namn. (”Save as”)
Ctrl+x s	Spara alla ändrade buffertar till filsystemet.
Ctrl+x i	Läs in en annan fil och infoga i bufferten vid markörens position.
Ctrl+x k	Döda bufferten som markören just nu befinner sig i. (”kill”, ”Close file”)
	Sök och ersätt
Meta+%	Sök och ersätt. Fråga för varje sak som skall bytas om den skall bytas eller ej, svara med **y** eller **n**. Avbryt kommandot med **ENTER**eller piltangenterna.
Meta+x replace-string	Sök och ersätt ovillkorligen utan att fråga om lov.
Meta+x query-replace-regexp	Sök och ersätt med reguljära uttryck.
Meta+x replace-regexp	Sök och ersätt med reguljära uttryck utan att be om lov vid ersättning.

	Markering, radering och kopiering av text
Ctrl+k	Radera innehållet på aktuell rad från markören och framåt.
Ctrl+k Ctrl+k	Radera raden och radslutet helt, ta bort en rad helt ur bufferten.
Ctrl+mellanslag	Markera början av ett stycke från den plats där markören befinner sig. Flytta sedan pilarna i bufferten för att markera mer text uppåt eller nedåt.
Ctrl+y	Klistra in det du sist tog bort på den plats där markören just nu befinner sig. (Vanligaste sättet att flytta saker är att ta bort dem med något av ovanstående och sedan klistra in dem med detta kommando.)
	Bufferthantering
Ctrl+x b	Byt till en annan buffert (bra för att t.ex. växla fram och tillbaka mellan två buffertar, den senast använda är standardalternativ).
Ctrl+x 1	Det fönster där markören befinner sig blir det enda aktiva EMACS-fönstret.
Ctrl+x 2	Dela fönstret du står i i två delar i vertikalled så att två buffertar kan visas samtidigt ovanpå varandra.
Ctrl+x 2	Dela fönstret i två delar i horisontalled så att två buffertar kan visas samtidigt vid sidan av varandra.
Ctrl+x o	Hoppa med markören från det ena till det andra fönstret (”other”)
Ctrl+x Ctrl+b	Visa alla buffertar i olika fönster samtidigt.

Figur 2.19: De vanligaste kommandona i EMACS.

du nere **Ctrl** och hamrar upprepade gånger på _ kommer du snart att ångra dig igenom allt du gjort, för EMACS har ett gott långtidsminne.

Den andra kombinationen är **Ctrl+g**, vilket avbryter ett kommando som låser in dig i en minibuffer (understa raden) så att du inte kan ta dig därifrån. **Ctrl+g** kan även avbryta saker som t.ex. oavsiktliga ”avsluta”-kommandon (plötsligt kommer texten *Save file ...?* upp fast den inte borde). Dessa två kommandon är guld värda när du upplever att EMACS ”fastnat”.

När du editerar med EMACS sparas ibland säkerhetskopior av dina filer, vilka innehåller den gamla filen före senaste editeringen. Om filen du editerade hette *foo.txt* kommer säkerhetskopian att heta *foo.txt*∼. En del användare ogillar att det ligger sådana säkerhetskopior och skräpar och sitter därför ofta och plockar bort dem för hand med **rm** ***∼**, en ovana som *inte* rekommenderas, eftersom det är lätt att missa det avslutande ∼-tecknet.

Det sparas även med jämna mellanrum en fil som innehåller alla ändringar som gjorts sedan du öppnade filen. Denna heter typiskt *#foo.txt#*. Om EMACS kraschar eller något annat otäckt inträffar kan denna fil återskapa de ändringar som gjorts fram till senaste automatiska undansparningen. EMACS kommer automatiskt att tipsa om att du

kan återskapa (engelska: recover) filen nästa gång du editerar *foo.txt*.

EMACS kommer även med ett eget programspråk som minner om programmets ursprung — EMACS LISP eller ELISP, vilket är det språk som större delen av programmet är skrivet i, och som används av den som önskar skriva egna tillägg till EMACS. Som normal användare kommer du sällan i kontakt med ELISP, men ett ställe där det används är i inställningsfilen *.emacs* som ligger i din hemkatalog. Att göra inställningar i denna kräver viss insikt i ELISP, alternativt att du bara kopierar exempel som andra gjort åt dig.

JOE, PICO, NANO, MCEDIT etc.

Utöver de ”två stora” texteditorerna finns det, framför allt i Linux-system, flera enkla editorer. Som nämnts erbjuder GNOME och KDE sina egna editorer, men det finns också andra, som kan nämnas som hastigast. Om du tycker att VI och EMACS är alldeles för krångliga, kan det vara värt att prova någon av dessa.

En varning skall dock utfärdas: även om dessa verktyg råkar finnas i just ditt operativsystem, så är det ingen garanti för att det kommer att finnas i alla andra POSIX-system, och allra minst att det skulle finnas förinstallerat. Därför är det ofta nödvändigt att även kunna behärska VI och/eller EMACS.

mcedit är en editor som hör till filhanteringsprogrammet *Midnight Commander* (startas med **mc**). Midnight Commander är skrivet av Miguel de Icaza som senare kom att starta projektet GNOME. Såväl **mcedit** som hela Midnight Commander är bekant för den som tidigare i livet använt DOS-programmet Norton Commander. Om du tycker om detta verktyg så är det en lysande idé att fortsätta använda såväl Midnight Commander som **mcedit** under POSIX.

pico är en editor som hör till programmet **pine**[48] som egentligen är till för att läsa elektronisk post och nyhetsgrupper. Editorn manövreras med olika **Ctrl+**-kombinationer som pedagogiskt nog är uppradade längst ned på skärmen så att du slipper komma ihåg dem i huvudet. Det lurar en fara i Pico: det är gjort för att editera textfiler som skall läsas av människor och kommer därför att infoga radbrytningar och avstavningar där den finner det lämpligt. Det går att gå runt detta, men det orsakar mest problem. Pico kan därför inte rekommenderas för editering av systemfiler, inställningar och liknande. GNU-projektet som inte är speciellt förtjusta i den dis-

[48]Se avsnitt 11.3 på sidan 390

tributionspolicy som gäller **pico** har istället skapat en egen klon som heter **nano** och som fungerar i princip exakt som **pico**.

jed är en menybaserad texteditor som är liten, smidig, och imiterar funktionen hos andra editorer, främst EMACS.

joe är ytterligare en texteditor som förut var mycket populär bland de som inte orkade eller ville lära sig vare sig VI eller EMACS. Denna kan efter begäran fylla övre halvan av skärmen med en lathund som gör det lätt att lära sig kommandona, som samtliga är beroende av **Ctrl**-tangenten. Dessa kommandon har inget gemensamt med vare sig VI eller EMACS.

2.3.8 Datum och tid

Det händer inte så sällan att klockan i din dator går fel, eller att du vill veta vad klockan är, samt vilken dag i månaden det är. Av denna anledning finns kommandon för att visa och justera datum och tid i alla POSIX-system.

cal ger en kalender för ett visst år, t.ex. get **cal 2004** en kalender för år 2004. En egenhet för detta kommando är att det kan ge olika resultat för år 1752 och 1753 då vi bytte från juliansk till gregoriansk kalender. GNU-versionen av kommandot (**gcal**) tar hänsyn till de svenska specialiteterna och visar dessa år korrekt, om rätt parametrar ges.

date ger datum och tid. Detta används för att ta reda på vad klockan är, genom att bara skriva **date**.

Därutöver används **date** också för att *ställa* klockan. För att ställa klockan skriver du **date MMDDhhmm** där *MM* är månad, *DD* är dagen *hh* är timmen på dygnet och *mm* minuten på timmen. Observera att alla parametrar skall anges tvåställigt, d.v.s. med inledande nolla om så behövs.

Behövs mer noggrann inställning kan kommandot ta flera parametrar. **date MMDDhhmmCCYY.ss** ställer dessutom klockan till århundradet *CC*, året *YY* och sekunden *ss* på minuten, t.ex. **date 072421042003.00**.

time har däremot inget med datum och tid att göra i vanlig mening! Detta kommando används för att ta tiden på hur lång tid ett visst kommando tar att köra, som med en kronometer.

Beträffande datum och tid, se även avsnitt 9.6 på sidan 335 om NTP.

2.3.9 Rör, filter och skript

> …välj först den fil du vill kopiera och skriv sedan chmod iff /dev/null ; dc. dc ska sedan pipe:a till bg print $1 -la $PARAM exec tempfile trap. Processen kommer här att fork:a och child kommer sedan att gå vidare till rm -f optparse rmm typeset l1 -R.current scanf grep .sh -DSLACK, sen fork:ar processen igen och skickar resultatet i en pipe som du sedan kopplar till den fil som du vill kopiera.
>
> — Fejkad distanskurs i UNIX, allmänt spridd på Internet

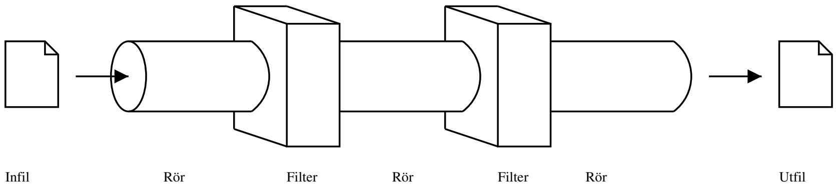
I slutet av 1960-talet avsatte försvarsorganisationen NATO stora resurser för att lösa ett problem som kallades *mjukvarukrisen*. Med detta menades att stora mjukvarusystem — exempelvis operativsystem, men också administrativa program och simuleringsprogram — hade blivit omöjliga att hålla samman. Programmen utvecklades till en gröt som inte gick att underhålla. Ett annat problem var att utvecklingsarbetet inte gick att parallellisera, det var svårt att få olika delar av ett arbetslag att tillverka varsin liten del av ett större projekt.

Vid en konferens i Garmisch i Tyskland som sponsrats av NATO lade Douglas McIlroy fram ett förslag på lösning av mjukvarukrisen som gick ut på att datorprogram skulle skrivas som komponenter som fogades samman på samma vis som elektroniska komponenter i en elektrisk krets. McIlroy var ”tredje mannen” bakom UNIX-utvecklingen på AT&T Bell Labs (jämte Ken Thompson och Dennis Ritchie), och införde därför detta tänkande i UNIX. Den ström som flyter i en elektrisk krets ersattes med strömmar av bytes, vilket till exempel kunde vara text.[49]

Genom att leda sådana strömmar genom tänkta *rör* (engelska: *pipe*) och genom att bearbeta dem i olika *filter*, kan ett datorprograms bearbetning av information liknas vid flödet i en krets eller ett vattensystem, där rören tillhandahålls av operativsystemet och filtren är olika självständiga datorprogram som bearbetar strömmen. Rören är i själva verket s.k. FIFO-köer, som vi ju stiftat bekantskap med tidigare; det som först går in i ena änden av ett rör kommer först ut i andra änden, som i figur 2.20.

När flera rör och filter fogas samman bildas en *rörledning* (engelska: *pipeline*). Den följd av tecken som skickas genom en rörledning kallas

[49] McIlroys idéer har sedan spridit sig till en mycket vid krets, *pipes and filters* är numera ett vedertaget s.k. *designmönster*, och den grundläggande tanken med mjukvarukomponenter är densamma för exempelvis CORBA, COM, o.s.v. Miguel de Icaza[13] menar att detta system med rör och filter är föråldrat och bör ersättas med GNOME-systemets Bonobo, en CORBA-baserad arktikektur.



Figur 2.20: Rör och filter processerar strömmar av information. Somliga kallar detta för *källa-flöde-vask* efter engelskans *source-flow-sink*.

för en *ström*. (Det är en ström av tal mellan 0 och 255 som avses, men dessa tal utgör oftast tecken och skickas radvis.)

Från dessa rör och filter, där varje filter är en egen process, ett självständigt datorprogram, härstammar den s.k. UNIX-filosofin om att skapa *små* program som gör *en sak* men som gör det *bra*, för att dessa skall kunna kopplas samman för att få en väl fungerande helhet.

Att använda rör och filter i POSIX-system är möjligt tack vare tre inbyggda ”specialfiler” i skalet. För varje nytt skal du öppnar skapas tre nya sådana unika filer.[50]

Dessa specialfiler upplevs som vanliga filer av alla datorprogram, men existerar inte i filsystemet i egentlig mening, d.v.s. de finns inte i filträdet. Dessa är:

stdin utläses *standard input* och är den fil där ett kommando hämtar sin indata, om inget annat angivits. Om du startar ett kommando, exempelvis **sh**, så kommer tecken att läsas från *stdin*. Om inget rör är kopplat till stdin, kommer programmet att anslutas till standardterminalen, det fönster där du startade kommandot, och kommandot kommer att börja filtrera det som kommer från tangenterna. Det är på detta vis skalet tar hand om tangenttryckningarna i en terminal.

stdout utläses *standard output* och är den fil där ett kommando skriver sin utdata. Om du startar ett kommando utan att ansluta något rör till *stdout*, kommer denna fil automatiskt att anslutas till standardterminalen, och på så vis kan du både skriva kommandon på *stdin* och ta emot resultat från *stdout* i exempelvis ett skal. Du uppfattar kanske aldrig att det är fråga om två filer, tangentbordet anslutet med ett rör till *stdin* och skärmen ansluten med ett rör till *stdout*, men det är faktiskt exakt så det hela fungerar.

[50] D.v.s. dessa filer följer med skalet, hör samman med skalet, inte med operativsystemet eller terminalen.

Symbol	**Namn**	**Funktion**
|	"pipe"	kopplar stdout från kommandot till vänster om tecknet till *stdin* på kommandot som står till höger om tecknet. Kommandon grupperade med | kallas för en *rörledning* (engelska: pipeline). Många rörstumpar ger alltså en rörledning.
<	"mindre än"	Om det som står till höger om tecknet är en fil i filsystemet, kopplas denna till *stdin* för kommandot närmast till vänster.
>	"större än"	Om det som står till höger om tecknet är en fil i filsystemet, kopplas denna till *stdout*. Till vänster måste det finnas ett kommando med någon form av utmatning till *stdout*. Om filen redan existerar kommer innehållet att skrivas över, om den inte existerar skapas en helt ny fil med angivet namn.
>>	"mycket större än"	fungerar som > men lägger till resultatet i slutet av filen om den redan existerar, annars skapas en ny fil.
n>&m	"anslut fil"	ansluter filbeskrivare n till filen som just nu används av filbeskrivare m. Det vanligaste (enda?) användningsområdet för denna operation är sekvensen **2>&1** vilket ansluter *stderr* till *stdout* så att programutskrifter och felutskrifter blandas samman till *en* fil, som i **grep foo * 2>&1 > foo.txt**
n>	"anslut fil"	ansluter filbeskrivare n till filen som står till höger om >. Den vanligaste användningen är att skicka *stderr* till en egen fil med t.ex. **grep foo * > foo.txt 2> err.txt**.

Figur 2.21: De vanligaste tecknen för omdirigering av strömmarna i rörledningar.

stderr utläses *standard error* och är avsedd för felmeddelanden. Denna ansluts nästan alltid till standardterminalen: även om *stdin* och *stdout* är anslutna till två filer i filsystemet istället för att vara anslutna till terminalen, vill du ofta direkt se om och när något går fel i kommandot.

Till varje specialfil hör en filbeskrivare (engelska: file descriptor) vilket är ett nummer. I själva verket har alla filer operativsystemet använder sådana nummer, men en vanlig användare som inte är programmerare behöver sällan använda eller bekymra sig om dem. *stdin*, *stdout* och *stderr* har filbeskrivarna 0, 1 och 2. De andra filbeskrivarna som används av skalet börjar sedan från 3 och uppåt. Detta dyker upp i en del specialfunktioner nedan och kan därför vara bra att känna till.

Rör och filter använder några speciella tecken för att indikera hur dessa "specialfiler" skall koppas samman. Dessa återfinns i figur 2.21.

De rörledningar som åstadkoms med dessa teckensekvenser kan sedan grupperas i olika *listor*, vilket vi sett exempel på redan tidigare. Dessa återfinns i figur 2.22.

;	synkron lista	semikolon kommer att separera två rörledningar (d.v.s. enstaka kommandon eller hela sekvenser avskilda med |) och utföra kommandona i rörledningen till vänster först, och sedan rörledningen till höger. ”gör först det till vänster och sedan det till höger om ;”.
&	asynkron lista	&-tecknet kommer att separerera en rörledning som står till vänster om &-tecknet och utföra denna asynkront, dvs parallellt med allt som följer efteråt. Detta känner vi igen som sättet att starta ett bakgrundsjobb, och givetvis är detta en och samma sak.
&&	lista med villkor	fungerar precis som en vanlig lista, men rörledningen som följer till höger utförs bara om den till vänster lyckades. En ;-lista utförs däremot ovillkorligen.

Figur 2.22: De vanligaste tecken för att åstadkomma kommandolistor med rörledningar.

Med dessa kunskaper och några vanliga hjälpkommandon är det lätt att bygga rörledningar som utför komplexa saker. Kommandona agerar ofta filter i rörledningarna, och jobbar vanligtvis på en textrad i taget. Eftersom POSIX-systemen normalt håller sig till textfiler är detta naturligt.

awk (vilket är namngivet efter ursprungsförfattarnas initialer: Al Aho, Peter Weinberger och Brian Kernighan.) är egentligen ett helt eget litet programspråk. Hela program kan tas som inparameter i form av en fil, men när rör- och filterkonstruktioner görs med **awk** skickas själva programmet oftast med i själva kommandot: **awk '{print $2}' < foo.txt** kommer att skriva ut andra kolumnen på varje rad i filen *foo.txt* förutsatt att kolumnerna är separerade med mellanslag eller tabbar. Ett annat sätt att beskriva samma form av rörledning är naturligtvis **cat foo.txt | awk '{print $2}'**.

Om filen *foo.txt* exempelvis innehåller:

```
Erik Stemme
Gunnar Ehrling
Viggo Wentzel
```

Kommer resultatet att bli en lista med efternamn. Ett lite mer komplicerat exempel:

```
awk 'BEGIN{FS=":"}{print $1}' < /etc/passwd
```

Detta skriver ut namnen på alla användare i operativsystemet. Först sätts fältseparatorn till ”:”, därefter skrivs första kolumnen

ut. Om du jämför med passwd-filens utseende på sidan 48 inser du nog hur programmet fungerar.

Detta program kan förenklas med växeln **-F** eftersom det är vanligt att byta just fältseparatorer. **awk -F: '{print $1}' < /etc/passwd** gör samma sak. Alternativt kan du skriva programmet (**BEGIN ...**) i en helt egen fil med namnet *prog.awk* och köra det med exempelvis **cat /etc/passwd | awk -f prog.awk**. Rör och filter kombinerat med olika kommandoparametrar ger en uppsjö olika möjligheter att åstadkomma samma resultat, vilket är något av tjusningen.

För att fullständigt lära sig språket **awk** krävs en del, och du får läsa manualsidan eller någon speciell bok i ämnet om du vill lära dig mer.

cat har vi stött på tidigare, men nu kan vi prova att använda kommandot till att verkligen konkatenera, d.v.s. slå samman filer. **cat fil1.txt fil2.txt fil3.txt > foo.txt** löser problemet. Ett annat sätt att göra samma sak borde vara att skriva **cp fil1.txt foo.txt; cat fil2.txt >> foo.txt ; cat fil3.txt >> foo.txt** och det finns fler tänkbara varianter. Ett viktigt användningsområde för **cat** är att kopiera innehållet i en fil till *stdout*, så att resultatet sedan kan skickas vidare i en rörledning, exempelvis **cat foo.txt | grep bar** som kommer att skriva ut alla rader i filen *foo.txt* som innehåller ordet *bar*.

echo har vi också stött på tidigare. Då nöjde vi oss med att konstatera att kommandot skickar ut parametern på terminalen. I själva verket skickar ett kommando som **echo "foo"** texten *foo* till *stdout* där den tas emot av skalet, som sedan skickar den vidare till terminalen, */dev/tty*.

Det vanligast sättet att utnyttja echo är genom omdirigering med rör. **echo "foo" > /dev/null** kastar ut strängen i datarymden, och **echo "foo" > /dev/tty** skickar den till terminalen. **echo "foo" > foo.txt** sparar texten i en fil. Ett annat vanligt användningsområde är som vi visat att undersöka miljövariabler, som **echo $TERM** till exempel. Vi skall behandla miljövariabler i nästa avsnitt.

grep och **egrep** är fina på att plocka ut rader ur en fil: exempelvis kan du räkna antalet rader i en fil som innehåller ordet *bar* med kommandot **grep bar foo.txt | wc -l**. (Se mer om **wc** nedan.)

Namnet **grep** betyder *global regular expression print*.[51] **egrep** är sam-

[51] Ursprungligen från kommandot till **ed** för att exekvera ett reguljärt uttryck över en textfil: g/re/p

ma kommando som **grep** med skillnaden att det kan hantera utökade (extended) reguljära uttryck.

Kommandot **grep** används så mycket att vana POSIX-användare snart säger att de ”greppar” i filer efter det ena eller andra. För att exempelvis hitta alla filer i ett filsystem som innehåller *foo* som en del av den fullständiga sökvägen, skriver du **find / | grep foo**, men du kan naturligtvis lika gärna leta i din egen hemkatalog med **find** ~ **| grep foo**, och otalet andra varianter.

grep kan använda alla former av reguljära uttryck, och ofta även den utökade varianten av reguljära uttryck: **grep ’f[no][o][rd]*’ foo.txt** kommer exempelvis att skriva ut alla rader som innehåller något av orden *foo* eller *fnord*. (Se avsnitt 2.3.6 om reguljära uttryck och tabellen på sidan 77 för mer om reguljära uttryck och utökade reguljära uttryck.)

more är ett finurligt textformateringskommando, liksom GNU-varianten **less**. Vill du se en fillista över hela ditt filsystem en skärmsida i taget finns inget bättre kommando än **find / | more**, och vill du lista filerna i din hemkatalog en sida i taget fungerar **ls -la** ~ **| more** utmärkt. (Mellanslag före och efter |-tecknet är egentligen inte alls nödvändigt.)

sed vilket utläses *stream editor* är ett mycket bra och vanligt filterprogram som används i många rörledningar. Det absolut vanligaste användningsområdet för **sed** är att utföra ”sök och ersätt”-operationer av typen ”byt A mot B”, som i ordbehandlarens ”sök och ersätt”. Vad som sker är att **sed** byter sekvenser i en rad som dyker upp på *stdin* mot andra sekvenser på den rad som sedan matas ut på *stdout*.

cat foo.txt | sed -e ”s/ab/cd/g” kommer att byta alla förekomster av strängen *ab* mot *cd* i filen *foo.txt*. Den sista bokstaven, *g*, betyder att du vill byta *globalt* — vill du bara byta första förekomsten på varje rad kan du skriva **s/ab/cd/1** istället. På samma vis byter en tvåa i sista positionen den andra förekomsten (om den finns) o.s.v.

Den sträng vi valde att sätta till *ab* är i själva verket ett grundläggande reguljärt uttryck. Exempelvis kommer **sed -e s/foo*/bar/g"< foo.txt** att byta alla förekomster av *fo*, *foo*, *fooo* o.s.v. mot *bar*.[52]

sort sorterar innehållet i en fil i bokstavsordning. Sorteringen sorterar från vänster till höger på alla tecken. Exempel: användarlista sorterad i bokstavsordning: **awk -F: ’{print $1}’ < /etc/passwd | sort**

[52]Detta är återigen ett bra tillfälle att repetera uppbyggnaden av reguljära uttryck i 2.3.6.

Observera att < kan fällas in i rörledningen på detta eleganta vis utan problem.

tr är mer eller mindre en specialvariant av **sed**: det byter ut enstaka tecken i en ström mot andra tecken. Detta utbyte sker positionsvis med parametrarna till kommandot, exempelvis byter **cat foo.txt | tr ab cd** ut alla förekomster av *a* mot *c* och alla *b* mot *d*.

uniq plockar bort dublettrader som kommer efter varandra i en fil. Om det står samma sak på två rader efter varandra tas den andra förekomsten bort. Exempel: **cat foo.txt | uniq**. Problem uppstår dock om du vill ta bort alla dubletter oavsett om de står efter varandra! Detta löses lämpligen genom att först filtrera strömmen genom **sort** med rörledningen **cat foo.txt | sort | uniq**. På så vis får du i den resulterande filen en sorterad, unik lista.

wc utläses *word count* och anger antalet rader, ord och tecken som en fil innehåller, om t.ex. **wc foo.txt** ger resultatet `15 12 109` betyder detta att filen *foo.txt* innehåller 15 rader, 12 ord och totalt 109 bytes. Även om **wc** kan jobba direkt på en fil är det lika lättanvänt i en rörledning: **cat foo.txt | grep fnord | wc -w** kommer exempelvis att skriva ut det sammanlagda antalet ord på de rader som innehåller ordet *fnord* i filen *foo.txt*. Möjliga växlar är **-w**, räkna ord, **-l** räkna antal rader, **-m** räkna antal tecken[53] och **-c** räkna bytes.

Med arkitekturen för rör och filter närmar sig användandet av skalet ett verkligt programmeringsspråk. När vi så låter ett antal kommandon samlade i en fil på datorns disk köras och återupprepas vid behov, har vi tagit steget fullt ut till ett verkligt programspråk. Detta kallas ett *skriptspråk*, vilket är namnet för språk som innehåller en rad kommandon till operativsystemet som i sin tur använder färdiga komponenter för att utföra dessa kommandon.

Förvisso är detta inget fullödigt utrustat programspråk som C, perl, python eller java, men det är tillräckligt för att täcka en avancerad användares direkta behov. Uppstår ytterligare behov är det naturliga steget att börja använda ett riktigt programspråk för att skriva egna datorprogram, och så långt dristar vi oss inte i denna bok.

Innan vi presenterar skript skall vi först titta på miljövariabler, som utgör de *variabler*, i betydelsen ”etiketterade informationsbehållare” som används i det enkla programspråk som ett skript utgör. De har också stor betydelse för skalets beteende.

[53] Med ”tecken” menas i detta fall allt som en dator uppfattar som tecken, även radbrytningar. Det är alltså inte samma sak som funktionen ”räkna tecken” i en ordbehandlare.

Miljövariabler

I skal av alla slag finns alltid ett antal miljövariabler definierade, och du kan om så önskas definiera nya själv. Miljövariabler kan sägas vara temporära lagringsutrymmen för små textfragment som skalet behöver för sin funktion. Vi kan exempelvis skapa en miljövariabel med kommandot:

```
FOO=bar
```

Detta kommando kommer att tilldela miljövariabeln **FOO** värdet **bar**. Vi kan titta på vår nya miljövariabel med kommandot **echo $FOO**. Detta kommer förhoppningsvis att skriva ut texten *bar* i skalet. Miljövariabler används till flera praktiska ting, och i synnerhet vissa enkla inställningar. Den mest grundläggande inställningen som lagras i en miljövariabel är sökvägen till olika kommandon i operativsystemet, **PATH**. Denna kan undersökas med kommandot **echo $PATH**. Som synes refereras alla miljövariabler med ett $-tecken.

Du kan se alla definierade miljövariabler genom att skriva kommandot **env** direkt följt av **ENTER**.

Vissa miljövariabler är tillgängliga för andra program (processer), som startas från skalet, andra är det inte. De är vad som kallas *globala* respektive *lokala* miljövariabler. Om ett annat program som startas skall kunna läsa av en miljövariabel som du satt måste den vara global, vilket normalt åstadkommes med kommandot **export**. Detta är något förvirrande och anledningen är att skalet även kan fungera som ett programspråk, och om du kör flera program skrivna i skalspråket, s.k. *skript* (se 2.3.9), så kommer dessa ganska snabbt att dränka hela miljön med stora mängder miljövariabler, om de inte kan hållas lokala. Därför fordras att du anger med **export** att en viss variabel *skall* exporteras, d.v.s. göras global, genom t.ex.:

```
export FOO=bar
```

Det är även möjligt att exportera en variabel efter att den definierats lokalt:

```
FOO=bar
export FOO
```

En lista över exporterade (globala) miljövariabler åstadkoms genom att skriva **export** utan argument. Ibland kommer skalet då att demonstrera en del interna detaljer, såsom att globala miljövariabler definieras genom att internt köra kommandot **declare -x FOO="bar"**. Detta är normalt inte av speciellt stort intresse.

Ur rent teknisk synpunkt är miljövariablerna en s.k. *symboltabell*: kort och gott en lagringsplats med en etikett, så att en viss textsträng[54] kan associeras med en viss annan textsträng på ett unikt vis. Några vanliga fördefinierade miljövariabler återfinns i figur 2.23, sidan 99.

Av speciellt intresse är t.ex. miljövariabeln **$LANG**, som används för att välja språk. För Sverige kan följande värden på den variablen vara speciellt intressanta:

Värde	**Språk**
sv_SE	Svenska, i Sverige, enligt teckenkodning ISO 8859-1.
sv_SE.UTF-8	Samma sak, men med Unicode-teckenkodning enligt UTF-8.
se_SE.UTF-8	Samiska i Sverige, med Unicode-stöd.
yi_SE.UTF-8	Jiddisch i Sverige, med Unicode-stöd.

Notera att språkkoden *se* används för samiska, medan den som landskod betecknar landet Sverige. För våra övriga officiella språk saknas i princip språkstöd under POSIX idag, men för romani chib borde språkkoden vara **rom_SE**,[55] meänkieli däremot har jag inte ens kunnat hitta en språkkod.[56] Du kan få en lista över tillgängliga **$LOCALE**-värden med kommandot **locale -a**.

Det finns en hel rad inställningar för ”lokalitet”, vilka alla visas om du skriver **locale** (utan parameter). T.ex. kan du ställa in vilken typ av tidformat och papper (USA använder ej A4) som skall användas på detta vis. Du kan ändra bara en viss sådan **$LC_FOO**-variabel, men om du ändrar **$LANG** så ändrar du dock allihop på en gång.

Vissa miljövariabler vill du kunna ställa in i ditt skal så att de alltid är satta på ett visst vis. Detta görs normalt i en speciell fil som läses in av skalet varje gång du loggar in. För Bourne Again Shell är det exempelvis den dolda filen **~/.bashrc**. För att ställa in standardeditorn till EMACS, språket till unicode-anpassad svenska i Sverige, och skriva ut ett glatt hälsningsmeddelande vid varje inloggning införs följande rader i .bashrc:

```
export EDITOR=emacs
export LANG=sv_SE.UTF-8
echo "Hejsan!"
```

[54] En *sträng* definieras som en ordnad sekvens av tecken.

[55] Baserat på att språkkoden i ISO 639-2 används istället för ISO 639-1.

[56] Se även sidan 251 för att se hur du väljer motsvarande tangentuppsättning i fönstersystemet *X*.

Namn	**Innehåll**
$$	Innehåller aktuellt processnummer för det skal eller skript där variablen används. Du kan på ett brutalt sätt slå ihjäl det skal där du själv befinner dig med kommandot **kill -9 $$** (om du nu har något emot **Ctrl+d**. . .)
$COLUMNS	Anger antal kolumner i terminalfönsret.
$EDITOR	Anger vilken texteditor som skall användas om något program behöver editera filer. Lämnas denna variabel blank antas automatiskt att texteditorn **vi** skall användas.
$HOME	Sökvägen till användarens hemkatalog. Om du skriver **cd** utan någon parameter, är det i denna katalog du hamnar.
$LANG	Vilket språk, land (lokalitet) och teckenuppsättning som används på den ort där datorn befinner sig. I Sverige skall denna normalt innehålla värdet `sv_SE` eller `sv_SE.UTF-8`. Formatet anger först lokalt språk med två gemener enligt standarden ISO 639-1 och landet med två versaler enligt ISO 3166. (Den senare används även för t.ex. toppdomäner på Internet och framför postnummer på vanliga brev.) Andra exempel: amerikansk engelska: en_US, egyptisk arabiska: ar_EG, etc.
$LINES	Anger antalet rader i terminalfönstret
$LD_LIBRARY_PATH	Sökvägen till dynamiska länkbibliotek för olika installerade program. Detaljerna runt detta varierar med operativsystem, se 5.4.5.
$LOGNAME	Anger användarnamnet för den som just nu är aktiv i detta skal. I flera skal är detta synonymt med miljövariabeln **$USER**. Ibland finns också ett verktyg som heter **whoami** som skall rapportera vad den för tillfället inloggade användaren heter. När du stöter på en dator som någon lämnat utan att logga ut ordentligt, kan du ta reda på vem denne var med hjälp av denna miljövariabel eller kommandot **whoami**.
$NLSPATH	Sökvägar till översättningstabeller för texten som ingår i olika program. Används för att kommandon och liknande skall ge svenska meddelanden istället för engelska, till exempel.
$PATH	Sökvägar till program som kan köras i skalet eller från ett skript. Sökvägarna är separerade med kolon. Om ett kommando inte kan hittas antingen i själva skalet eller i dessa sökvägar, får användaren antingen ett felmeddelande eller ett förslag på ett kommando med liknande namn.
$PWD	Innehåller aktuell katalog, d.v.s. ”vart jag befinner mig” i filsystemshierarkin. Denna variabel avläses ofta med kommandot **pwd** som i princip bara skriver ut denna miljövariabel.
$SHELL	Anger vilket skal som används, i form av hela sökvägen till skalprogrammet.
$TERM	Anger namnet på den terminal som används. Om du arbetar i ett vanligt fönster på en grafisk desktop kommer detta normalt att vara något i stil med *xterm*. Vid textläge i Linux heter terminalen helt enkelt *linux* o.s.v. Se vidare avsnitt 2.3.11 om terminalinloggning.

Figur 2.23: Några vanliga miljövariabler. Som användare kan du se alla satta miljövariabler med kommandot **env** eller enstaka variabler med kommandot **echo $VARIABEL**. Själva $-tecknet är inte en del av variabeln, utan den syntax som används för att åberopa den.

```
#!/bin/sh
if [ "$1" = "off" ]; then
        if [ "x$DISPLAY" == "x" ]; then
            echo -e "\33[11;0]"
        else
            xset b off
        fi
        echo "Beep turned off."
fi

if [ "$1" = "on" ]; then
        if [ "x$DISPLAY" == "x" ]; then
            echo -e "\33[10;750]\33[11;250]"
        else
            xset b on
        fi
        echo "Beep turned on."
fi

if [ "$1" = "" ]; then
        echo "Usage: $0 <on|off>"
fi
```

Figur 2.24: Skript som slår av/på pipandet i terminaler på de flesta system.

Skript

Efter ett tag kommer du att upptäcka att flera av dina kommandosekvenser blir vardagsmat. Du skriver samma sak om och om igen, och det är väl inte så jobbigt att skriva **ls -l | less**, det är ganska lätt att komma ihåg att ”om jag sätter ihop list-kommandot med GNU less, får jag innehållet i en katalog listat en skärmsida i taget”.

Men en jobbigare harang som till exempel sekvensen för att stänga av det irriterande blippandet från terminalen, kan se ut såhär: **echo -e ”\33[11;0]”**.[57] Detta är inte mycket längre, men innehåller en del magiska siffror som inte är så enkla att hålla i huvudet. Att slå på blippandet igen är minst lika komplicerat.

Av den anledningen skriver du skriptet i figur 2.24. Skript är en form av datorprogram, och har du skrivit datorprogram förut så känner du nog snart igen dig. Kommandona i skriptet ser bekanta ut: där finns **echo** och något annat som heter **xset**. Men en sak i sänder.

Översta raden i ett skript: **#!/bin/sh** är inget annat än en fullständig sökväg i filsystemet till det skal som skriptet skall köras i. När skrip-

[57]Den exakta innebörden av denna kommandosekvens lämnar vi åt sidan, nöj dig med att veta att den stänger av pipandet.

tet startas kommer det nämligen att köras i ett helt eget skal, som en helt egen process. Du kan mycket väl använda ett annat språk till dina skript: **#!/bin/awk** eller **#!/bin/ash** hade fungerat lika bra. Du kan lika gärna hitta ett mer komplext språk angivet på översta raden: **#!/usr/bin/perl** eller **#!/usr/bin/python** är lika tänkbara alternativ. Resten av filen kommer att tolkas som om den var skriven i det språk som anges på den här **#!**-raden. Språket anges genom att peka ut dess programfil, den fil i vilken språkets kommandotolk ligger lagrad. Så i det allra enklaste fallet är det samma sak som standardskalet: */bin/sh* är den plats där standardskalet ligger i de flesta POSIX-system.

Resten av programmet består av villkorssatser: ett uttryck som står inom **if [″a″ = ″b″]; then ... else ... fi** är villkorat: om *a* är samma sak som *b* kommer det som står efter ordet *then* att utföras, i annat fall kommer det som står mellan orden *else* och *fi* att utföras.

Miljövariabeln med namnet **$1** är den parameter som givits till skriptet. Om jag exempelvis kallar min programfil för *beep.sh* så kan programmet startas med **beep.sh on** eller **beep.sh off**. Beroende på vilket som skrivits kommer olika delar av skriptet att utföras, och om ingen parameter angetts kommer skriptet att skriva ut en liten hjälptext som upplyser om hur det används.

$DISPLAY är en miljövariabel som är satt till något om programmet startats under fönstersystemet *X*: i det fallet är sättet att stänga av blippandet ett annat, då används programet **xset**. De underliga siffersekvenserna i **echo**-satserna talar om för en någorlunda normal terminal att sluta respektive börja pipa. (Se även stycket lite längre fram om terminaler.)

Skript skrivna i *sh* eller *bash* är något svårhanterliga. Skriptet måste exempelvis skrivas exakt som i figuren: hakparenteserna efter **if**-satserna är löjligt känsliga för mellanslag, det är nödvändigt att smälla in ett extra **x** i satser som **if [″x$DISPLAY″ == ″x″];** av den enkla anledningen att skalet inte klarar av att hantera det specialfall som uppstår då **$DISPLAY** är en tom sträng, utan helt sonika kraschar.

Det svåraste för en nybörjare på skript brukar dock vara att över huvud taget komma igång. Det finns lite små detaljer att tänka på, därför följer en punktlista.

- Först vill du ha en plats att lägga dina fina skript på. Skapa en katalog i din hemkatalog som heter **bin** för att följa de inofficiella filsystemskonventionerna: detta är det vanligaste stället att ha dem på: **mkdir bin**.

- Sedan måste du se till att skalet kan hitta dina program för att köra dem. Som nämndes i avsnittet om miljövariabler så använder

skalet miljövariabeln **$PATH** till detta. Ett bra ställe att lägga till detta är i din ~**/.bashrc**-fil som körs varje gång du startar ditt skal. (Om du använder något annat skal heter filen något annat, exempelvis **.zshenv** för Z Shell.) Öppna **.bashrc** med en texteditor och skriv till:

```
export PATH="$PATH:$HOME/bin"
```

Detta kommer att lägga till sökvägen till din nya skriptkatalog i miljövariabeln **$PATH** varje gång du loggar in. För att ändringen ska "ta" är du tvungen att logga ut och sedan in igen. Gör det!

- Kolla att miljövariablen **$PATH** blev korrekt inställd med **echo $PATH**. Finns sökvägen till din hemkatalog med i variabeln? Annars: försök hitta felet. Om **$HOME** inte är satt kanske du måste använda fullständig sökväg till din hemkatalog.

- Nu kan systemet hitta dina skript. Gå in i skriptkatalogen med **cd ~/bin** och editera filen **foo.sh** med en texteditor. Skriv till exempel in något i stil med nedanstående och avsluta texteditorn. Du kan även prova det mer avancerade **beep.sh**-skriptet i figur 2.24, men det kan bli svårt att hitta felen i det om du inte skriver av det rätt!

```
#!/bin/sh
echo "Foo!"
```

- Ditt skript finns nu i skriptkatalogen, men det går inte att köra det! Varför? Naturligtvis är det filrättigheterna som spökar. Du har inte talat om för filsystemet att du har rätt att köra den här filen som ett program! **chmod +x foo.sh** löser problemet.

- Prova direkt att köra ditt skript med **./foo.sh**. Fungerar det? Denna egenhet med **./** före programnamnet är en fullständig sökväg — det krävs nämligen när programmet inte ligger i någon av de kataloger som finns angivna i **$PATH**.

- Nu kan du prova skriptet från vilken plats som helst i ditt operativsystem. Skriv bara **foo.sh**! Skriptet lokaliseras via **$PATH** och körs sedan. Nu är det bara att börja fylla din **bin**-katalog med roliga skript.

Filnamnsändelsen **.sh** som har använts på skripten här är bara en förtydligande sak. Du kan kalla filer för bara **foo** eller liknande: om du tittar på de ordinarie programfiler som normalt ligger i */bin* så ser du att de inte har några filnamnsändelser.

```
#!/bin/sh
for file in *.foo; do
        base=`basename $f .foo`
        mv $file $base.bar
done
```

Figur 2.25: Skript som kopierar alla filer med filnamn som slutar på **.foo** till samma namn men med ändelsen **.bar**.

Samma regler gäller för skript: är det en körbar fil så kan den köras, oavsett vad den heter. Är det inte en binär programfil, så kommer skalet att se om första raden innehåller uppgift om någon kommandotolk i stil med **#!/bin/sh** och sedan försöka köra programmet i den tolken.

Användare av andra operativsystem är kanske inställda på att körbara filer skall heta något i stil med **foo.exe** — detta gäller *inte* i POSIX-familjen. Apropås användare av andra operativsystem: i CP/M och MS-DOS finns en funktion för att kopiera alla filer som har en viss trebokstavs filändelse till en annan, med **move *.foo *.bar**. Detta fungerar inte i POSIX. Skriptet i figur 2.25 gör dock samma sak.

2.3.10 Nätverket

Nätverket är intimt förbundet med operativsystemen i POSIX-familjen. Detta beror på att när Internet i form av ARPANET en gång byggdes upp användes det för att få datorer som körde främst VMS och UNIX att samarbeta med varandra, och sedan dess har nätverksfunktioner varit en naturlig del av operativsystemen i POSIX-familjen, långt innan det kom till andra operativsystem. Flera POSIX-system kan inte ens fungera utan nätverk: även om ett fysiskt nätverkskort saknas i datorn krävs att operativsystemet ändå emulerar ett nätverk med en enda dator för att det skall fungera ordentligt.

Det är bland annat på grund av detta som alla datorer i POSIX-familjen har ett namn som måste ställas in vid installation. För det mesta ger kommandot **echo $HOSTNAME** eller **echo $HOST** information om vad datorn heter på nätverket.

Hur nätverket fungerar i praktiken och hur du ställer in det varierar mellan olika operativsystem. Linux har en relativt väletablerad standard som kommer att avhandlas i avsnitt 9.3.

Klient och server

Ett relativt vanligt begrepp i POSIX-världen är konceptet med klienter och servrar. (Ibland används ordet *värddator* istället för server, men detta språkbruk har inte slagit igenom.) Detta förutsätter i sin tur att datorn i fråga har någon form av nätverk installerat.

Tanken är denna: svåra, tunga och driftsäkra tjänster som alltid måste finnas tillgängliga skall skötas av *servrar*, vilket är datorer som är installerade, konfigurerade och utplacerade i syfte att tillhandahålla denna tjänst.

Med *tjänst* menas något som användare eller andra datorer har nytta av. World Wide Web-servrar, databaser av ena eller andra slaget, filservrar som tillhandahåller hela filsystem, epost-servrar som lagrar och skickar elektronisk post vidare, servrar som sköter telefonnät, elnät och koordination av SJs tågtrafik, biljettbokningssystem för biografer och resebyråer: alla är de tjänster. Gemensamt har de att vi blir arga när de inte fungerar.

När vi irriterar oss på att Dagens Nyheters hemsida eller Swebus Express resbokning inte fungerar är vi *klienter*. Vi är klienter som ansluter oss till servrar, ofta via webbläsare.

Att vara klient eller server är dock också en *roll*: den server som bokar våra biljetter kan i sin tur vara klient till en annan server som tillhandahåller adressen hem till dig från ett speciellt adressregister, så att biljetterna kan skickas till rätt ställe.

De tjänster som tillhandahålls av en server kommer i praktiken oftast från ett datorprogram, som kör i en process på servern. Processen är i normalfallet en demon. Demonen startar och stoppar andra hjälpprocesser, kommunicerar över nätverket med andra servrar och levererar begärda tjänster till klienter. Tjänsterna levereras ofta i form av strömmar av text: ett populärt exempel i dessa dagar är så kallade ”webbtjänster”: strömmar av text i standardformatet XML som bearbetas av en demon och skickas via port 80 på servern, över protokollet HTTP.[58]

Men webbtjänster och annat som glimmar guld kan vi skjuta åt sidan: den äldsta tjänsten på internet heter *telnet*. Denna kan startas med kommandot **telnet foo.bar.com** (förmodligen kommer namnet från något i stil med: ”teletypewriter over network”) och kommer då att försöka ansluta ditt terminalfönster till datorn på Internet med namnet *foo.bar.com*. Emellertid är det sällan en så bra idé att använda telnet i dessa dagar, de flesta anser detta system vara osäkert och föredrar att använda SSH som gör ungefär samma sak (**ssh foo.bar.com**), men med högre säkerhet, autentisering och kryptering.

[58] Denna namedropping är kanske lite stressande. Ta inte terminologin på för stort allvar.

Detta skall vi dock stifta bekantskap med vad det lider.

2.3.11 Terminalinloggning

> War is peace. Freedom is slavery. Ignorance is strength. Delete is backspace.
>
> — Travesti på George Orwell

I början av detta kapitel tittade vi på skal, och nämnde då att dessa var kopplade till en terminal. Vi har också fått veta att denna terminal symboliseras av en blockenhetsfil med namnet */dev/tty* och att den i normalfallet tar emot den information som skalet vill ha från *stdin* och skriver ut resultatet på *stdout*. Vi har nu också sett att POSIX-system är tätt integrerade med nätverk, och att nätverket gör det möjligt att logga in i en annan dator via ett fönster som vi kallar för en terminal.

Vi nämnde också att terminalerna var öppningar in i datorn, förr genom en sladd ansluten till en port på datorn, som sedan var kopplad till en skärm och ett tangentbord av enklaste typ. Dessa enkla terminaler kallas ibland *TTY*:er. Anledningen till att de heter så är att det är en förkortning av namnet *Teletypewriter*, detta var en så kallad terminalskrivmaskin[59] som kunde mata in och ut text på ett väldigt simpelt sätt. Ordet kom sedan att bli synonymt med terminal.

Alla program som körs inne i skalet tar en ström från *stdin* och producerar en ström på *stdout*. Skalet skickar sedan detta vidare till terminalen. Undersök gärna hur */dev/tty* fungerar: **echo "foo" > /dev/tty** gör att *foo* skrivs ut på din terminal, och följande "texteditor" kan vara värd att tillgripa när inget annat fungerar: **cat /dev/tty > foo.txt**. Avsluta sedan editeringen med **Ctrl+d** (filslut).

I våra dagar behöver terminalen inte vara någon speciell utrustning kopplad till datorn med en sladd. Den *kan* dock vara det, det är normalt inte så avancerat att ansluta en stenåldersterminal till en vanlig PC:s RS232-kontakt och faktiskt lyckas med att använda den. Mestadels använder vi i dag dock *virtuella terminaler*, i form av fönster som öppnats på datorn.[60] Virtuella terminaler kan också öppnas i en grafisk miljö, som fönstersystemet *X* (som vi kommer att avhandla i ett eget kapitel).

Alla POSIX-system kan på ett eller annat vis åstadkomma en terminal, det gäller bara att komma underfund med hur. I ett grafiskt system

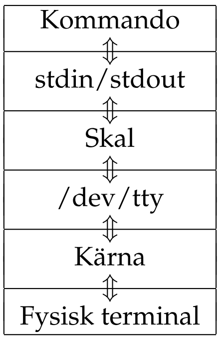
---

[59]Se vidare sidan 79

[60]Linux har exempelvis ofta flera sådana virtuella terminaler öppna som kan lokaliseras med **Ctrl+Alt+F1**, t.o.m. **F6**.

finns den oftast i någon meny, ibland undangömd för att inte i onödan presentera maskinens komplexitet för användaren.[61] Du känner igen terminalen på att den omedelbart startar ett skal för att du skall kunna använda den till något, vilket kan se ut som i figur 2.3.

Vi skall nu nämna något om hur terminalen samarbetar med operativsystemet. Eftersom en terminal ansluten med en sladd är en hårdvarusak som kärnan tar hand om, dyker den som sagt upp i */dev/tty*. Kärnan vet hur den skall bete sig för att rent elektriskt prata med terminalen. Om terminalen inte skulle vara något elektriskt, kommer den ändå att hamna som en enhet i */dev/tty*. Därutöver sätts miljövariablerna **$TERM**, **$LINES** och **$COLUMNS**. Dessa talar om för skalet vilka tecken som kan sändas till terminalen, samt hur många rader och kolumner terminalen har. Du kan undersöka din egen terminal med med **echo $TERM** och motsvarande för rader och kolumner. Ett vanligt Linux-system har exempelvis ofta en virtuell terminal med *$TERM=xterm* (i fönstersystemet X) eller *$TERM=linux* (konsollen).



Ett namn som *xterm* eller *linux* ger dock inte operativsystemet någon information om hur dessa terminaler egentligen fungerar — hur flyttas exempelvis markören runt? Hur byter texten färg? Vilka färger finns, eller är terminalen svart/vit? Sådana saker måste registreras undan speciellt för att skalet och andra program skall kunna komma åt att skriva ut och läsa text från terminalen. Här skiljer sig olika operativsystem åt, men de allra flesta använder endera av två databaser: *terminfo* och *termcap*. Dessa är på intet vis delar av POSIX-standarden.

termcap är det gamla systemet för att hålla reda på terminalernas egenskaper. I en fil som på de flesta system heter */etc/termcap* lagras alla terminalers egenskaper. Denna levereras med operativsystemet, och enskilda användare kan lägga egna *termcap*-filer i sina

[61] Till och med MacOS X, som tidigare vägrade inkludera alla form av kommandoradsverktyg, har numera en virtuell terminal.

hemkataloger för att lagra andra terminalers egenskaper. root-användaren får editera */etc/termcap* för att lägga till nya terminaler.

terminfo är ett nyare system för att hålla reda på terminalernas egenskaper. I de system som använder terminfo lagras terminalernas egenskaper i katalogen */usr/share/terminfo* och sedan genereras */etc/termcap* automatiskt utifrån dessa filer. Med hjälp av kommandot **tic** (*terminfo compiler*) kan nya terminaler läggas till i denna databas, eller i en lokal databas som är unik för varje användare och utgör dennes "egna" terminalbeskrivningar.

Om du har en felaktigt angiven terminaltyp eller en terminal som programmet du använder inte känner till kan konstigheter uppstå. Exempelvis kan VI få för sig att du har en terminalskrivmaskin[62] (d.v.s. sämsta tänkbara terminal) och således bara tillåta dig att editera en rad i taget.

Ett annat vanligt fel är att all text blir inverterad. Detta beror på att alla terminaler som kallar sig *xterm* inte har samma egenskaper: *xterm* för Solaris är t.ex. helt monokrom, medan *xterm* i GNU/Linux kan hantera färg.

Om du misstänker att din terminal är felinställd är det en bra början att kolla om den är inställd över huvud taget med **echo $TERM**. Om denna variabel är blank har du hittat felet. Prova då några vanliga terminaltyper, exempelvis med **export TERM=vt100**, en mycket vanlig terminal, som används mycket för telnet, SSH och liknande.[63] I Linux är det oftast **linux** (för textfönster i konsolläge) eller **xterm** (i fönstersystemet *X*) som gäller som terminaltyp.

2.3.12 At- batch- och cronjobb

På POSIX-system skall det standardmässigt finnas två stycken demoner, *atd* och *crond*. Dessa används för att schemalägga kommandon för enstaka eller regelbundna körningar. Dessa kallas *jobb* med härledning från att processer som kontrolleras av ett skal kallas för jobb.

Atjobb kallas jobb som utförs av *atd* och är av typen "utför detta kommando, men inte nu utan vid tidpunkten *t*". Vid en vald tidpunkt kommer *atd* att starta ett skal och köra ett visst kommando.

Cronjobb kallas jobb som utförs av *crond* och är av typen "utför detta kommando regelbundet", exempelvis "gör detta dagligen" eller

[62] Se sid 79

[63] För mer information om telnet och SSH, se sidan 328.

”gör detta en gång i månaden”. Flera POSIX-system använder sig själva av schemalagda cronjobb för att sköta visst systemunderhåll.

Kommandot **at** som finns i POSIX-standarden är egensinnigt implementerat på olika plattformar och följer sällan standarden. På Linux finns en uppsättning kommandon: **at** för att lägga till atjobb, **atq** för att lista dem, **atrm** för att ta bort dem. Något som däremot följer standard är kommandot **batch**, som schemalägger ett kommando för att utföras *nu* (eller ”typ nu”), men i ett separat skal. Om du vill slippa trixa omkring med jobbkontroll och **disown** för att släppa dina jobb på grönbete, är **batch** ett utmärkt kommando.

Såväl **at** som **batch** är interaktiva kommandon. De tar instruktioner direkt när du startar dem. Såhär kan en **batch**-session se ut:

```
# batch
at> find / > batch.txt
at> <EOT>
job 9 at 2003-07-27 23:48
```

För att slippa ifrån **at**-prompten tryckte jag **Ctrl+d** vilket betyder filslut. Om jag ångrar mig kan jag i vanlig ordning trycka **Ctrl+c** innan jobbet startats. Skillnaden med **at**-kommandot är att du kan ange en tidsangivelse som parameter så att programmet startar vid ett senare tillfälle. Du kan schemalägga jobb flera år in i framtiden om så önskas.

Cronjobb skiljer sig från atjobb genom att de är periodiska. De upprepas med vissa intervall. Typiska cronjobb är rotering av loggar: systemet använder cronjobb för att spara undan gamla loggfiler (noteringar av systemhändelser) med jämna mellanrum. När de legat och skräpat tillräckligt länge slängs de.

Du kan använda dessa funktioner till exempel för att göra backupkopior av din hemkatalog med jämna mellanrum, eller diverse andra saker. Det typiska för cronjobb är att de utförs när du inte är inloggad, så förvänta dig inga direkta meddelanden av vad som sker, den gamla hederliga *stdout* brukar i allmänhet skickas som ett brev till dig, men har du inte ställt in funktioner för elektronisk post på din dator kommer det brevet aldrig fram. Istället hamnar det i någon buffert i */var/spool/mail/root*[64] eller liknande, där det stannar i tid och evighet.

Cronjobb startas från en så kallad *crontabell*. Denna innehåller ett kommando per rad, och de sex första positionerna symboliserar tidpunkter då jobbet skall köras. Därefter följer själva kommandot. Alla

[64] Om du inte har fungerande epost på din dator finns det anledning att då och då titta om det ligger en stor fil som heter *root* i katalogen */var/spool/mail*. Där hamnar ibland lite allt möjligt.

Crontabellens format

Minut	Timma	Dag	Månad	Veckodag	Kommando
0-59	0-23	1-31	1-12	0-7	...
Exempel 1: Kör en gång i månaden vid midnatt					
0	0	1	*	*	foo
Exempel 2: Kör varje måndag klocka 8:00					
0	8	*	*	1	foo
Exempel 3: Kör var 10:e minut, alltid					
*/10	*	*	*	*	foo
Exempel 4: Kör en gång i kvartalet, kl 15:00					
0	15	1	1,4,7,10	*	foo

Figur 2.26: Crontabellens fält. Intervall kan anges med komma för att räkna upp flera 1,2,3,. . . - (bindstreck) för intervall, 1-5 eller * för att beteckna *alla*. Efter en asterisk kan även ett intervall anges som i */2 — *varannan*.

cronjobb hanteras med kommandot **crontab**. Det enda **crontab**-kommandot egentligen gör är att editera en fil som underhålls av operativsystemet, och där det håller reda på de cronjobb som finns och när de skall köras.

crontab -l listar crontabell så du ser hur den ser ut just nu. Normalt är den naturligtvis tom, och det är inte många användare som använder cronjobb.

crontab -r raderar crontabellen och dödar alla framtida cronjobb.

crontab -e editerar crontabellen med den editor som du har ställt in i miljövariabeln **$EDITOR**. Har du inte ändrat denna är det VI du möter.

Du kan normalt få full information om det format crontabellen använder med kommandot **man 5 crontab**, men utseendet kan vara något åt det här hållet:

```
0 23 * * * foo
```

Denna crontabell säger att kommandot *foo* skall köras klockan 23:00 varje dag. Fälten är uppdelade så som framgår av figur 2.26.

Du kan ha kommentarer i crontabellen genom att inleda raderna med #, och ange att ett speciell skal skall användas genom att på en

ensam rad skriva **SHELL=/bin/ash** till exempel. Standardskalet för användaren kommer att användas av både at- och cronjobben om inget annat anges.

KAPITEL 3

De fria mjukvaruprojekten

> What sphinx of cement and aluminum bashed open their skulls and ate up their brains and imagination?
>
> Moloch! Solitude! Filth! Ugliness! Ashcans and unobtainable dollars! Children screaming under the stairways! Boys sobbing in armies! Old men weeping in the parks!
>
> — Allen Ginsberg

De två viktigaste mjukvaruprojekten i denna framställning är GNU och Linux. Dessa kommer att behandlas utförligt i kapitel 5.

Förutom GNU och Linux hämtar ett GNU/Linux-system många av sina vitala komponenter från flera andra projekt. Detta kan som nämnts uppfattas förvirrande om du är van vid att ett operativsystem levereras färdigförpackat i en kartong. Emellertid ger denna organisation av autonoma projekt en oerhörd potential till diversifiering och påverkan utifrån. Detta kapitel avhandlar de fria projektens organisation och karakteristika jämfört med den organisation som kännetecknar s.k. "traditionella" mjukvaruföretag.

Detta kapitel avser endast att ge en kort introduktion till de fria mjukvaruprojektens dynamik och motivationer. Detta är ett kärt diskussionsämne som alltför ofta diskuteras på en låg nivå, men som fascinerar alla.

Ingen vet egentligen varför det tillverkas fria mjukvaror: ju mer du studerar de fria projekten i mikroskop och försöker förstå dem, ju mer

undflyr de betraktarens blick. Alltför många var alltför tidiga att helt döma ut GNU/Linux som ”omöjligt”, ”hopplöst” o.s.v. under de tidiga dagarna. Men dessa människor refererade hela tiden till omöjligheten att leva på att utveckla system som detta, och refererade till det hopplöst dumma i att sätta sig mot marknadskrafternas dynamik. Detta ingick kort sagt inte i deras uppfattning om verklighetens beskaffenhet.

Nu är det annorlunda. Alla vet att fria mjukvaror finns, blomstrar och fungerar, att de har haft stor ekonomisk betydelse. Kanske inte ännu på skrivbordet, men på miljontals andra ställen.

GNU-projektet (se avsnitt 5.1) har definerat termen *fri mjukvara* som används ganska mycket i den här boken. Den engelska termen *free software* har dock vållat en hel del problem i världen. Oavsett vad Richard Stallman menade med detta begrepp i begynnelsen, så har Free Software Foundation gjort fullständigt klart i sin defintion av fri mjukvara hur detta skall tolkas[11]:

> “Free software” is a matter of liberty, not price. To understand the concept, you should think of “free” as in “free speech,” not as in “free beer.”

Free Software Foundation har även lagt sig till med devisen *Free as in Freedom* (plockad från titeln på Richard Stallmans biografi[39]) för att ytterligare understryka detta. Var vänlig och gör aldrig misstaget att tro att *fri* i detta sammanhang betyder *gratis*, *antikapitalistisk*, eller något annat dumt. Defintionen fortsätter[11]:

> Fri programvara handlar om en användares frihet att köra, kopiera, distribuera, studera, ändra och förbättra programvaran. Mer precist så handlar det om fyra sorters frihet för användaren av programvaran:
>
> - Friheten att köra programmet, för ett godtyckligt syfte (frihet 0).[1]
> - Friheten att studera hur programmet fungerar och att anpassa det för sina behov (frihet 1). Tillgång till källkoden är ett villkor för detta.

[1]Detta gäller i allmänhet inte proprietära mjukvaror, som ofta åtföljs av en s.k. EULA (End-User License Agreement) som verkligen begränsar vad användaren får göra med programmet, t.ex. förbjuds ofta användning av programmet i brottsliga syften, såsom att använda ett bildbehandlingsprogram för att kopiera sedlar. Även om det kan argumenteras för och emot sådana avtal, finns det inget i de fria mjukvarulicenserna som begränsar användandet på detta vis, t.ex. måste inte en slutanvändare ”godkänna” GNU GPL, däremot ger GNU GPL slutanvändaren en rad rättigheter, vilket inte är samma sak.

- Friheten att vidaredistribuera kopior så att användaren kan hjälpa sin nästa (frihet 2).
- Friheten att förbättra programmet och att ge sina förbättringar till allmänheten så att hela samhället drar nytta av dem (frihet 3). Tillgång till källkoden är ett villkor för detta.

När vi talar om fri mjukvara, är det denna definition som gäller. Liknande begrepp som inte skall förväxlas med detta är t.ex.:

public domain-mjukvara inkluderar kategorin fri mjukvara, men behöver inte betyda att mjukvaran är fri i ovanstående bemärkelse (t.ex. saknas ofta källkoden).

shareware-mjukvara är i allmänhet program som kräver att du betalar något till författaren, för att få fullständig tillgång till ett helt fungerande program. Detta i sig är inte oförenligt med att mjukvaran är fri: tillverkaren kan exempelvis ge dig gratis tillgång till ett kompilerat, körbart program med begränsad funktionalitet, men däremot *enbart* tillhandahålla källkoden till det fullständiga, och fullt fungerande programmet. (Som då kan köpas färdigkompilerat.) Dock uppträder termen oftast i ett sammanhang där de ovanstående friheterna inte är med, och scenariot är så vitt jag vet helt teoretiskt.

3.1 Ekonomisk förklaring

I en artikel med titeln *There Is No Open Source Community*[2] har John Mark Walker på ett utmärkt sätt summerat den ekonomiska infallsvinkeln på begreppet fri mjukvara och öppen källkod. Den grundläggande tes han driver är förmodligen identiskt med många av de tankegångar som förekommer på företag som engagerar sig i fri mjukvara. Som bekant är det, i varje fall i teorin, de ekonomiska argumenten för och emot ett visst system som till sist och ytterst avgör om ett vinstdrivande företag skall handla på det ena eller andra viset. Ett företag kommer, med rätt information och rätt bedömning av kostnader, risker, för- och nackdelar att använda fri mjukvara om detta på det hela bedöms bli mer lönsamt än att gå till de proprietära alternativen.

[2]Se Walker[37], en uppföljande artikel från ett s.k. ”demand-pull”-perspektiv har ännu inte publicerats i skrivande stund. Titeln är något missvisande (och antagligen avsiktligt provocerande), den betyder just bara att det inte finns *en* rörelse för öppen källkod, men det kan mycket väl finnas *många*, sinsemellan olika.

Walker menar att den nuvarande beskrivningen av fria mjukvaror och öppen källkod felaktigt fokuserar på en individualistisk berättelse: det är berättelsen om personer som Richard Stallman och Linus Torvalds, och i förlängningen en berättelse om en social rörelse.

Han hävdar också att det skulle vara frågan om en "David-mot-Goliat berättelse"[3] om hur de små gräsrötterna och småföretagen bekämpar de ondskefulla mjukvarumonopolen. På det hela taget ogillar han detta berättande och menar att det är bättre att hänföra den fria mjukvarans framgångar på rent ekonomiska faktorer.[4]

Han presenterar därför ett resonemang i helt annorlunda termer, som kan ställas upp i punktform:

- Fri mjukvara agerar på samma marknad som all annan mjukvara och drivs av tillgång och efterfrågan.

- Mjukvarumarknaden har gått från att vara smal och vertikal till att vara bred och horisontell.

- Internet har gjort det möjligt att producera, paketera och distribuera mjukvara till en oerhört låg kostnad jämfört med tidigare system: disketter, CD-skivor, kassettband etc som skickades med vanlig postgång. Fri mjukvara använder nästan uteslutande denna produktionsmiljö och distributionskanal vilker drar ned produktionskostnaderna till ett minimum. Denna form av distribution till användarna är dessutom snabbare, vilket ger det ännu en fördel över de proprietära mjukvarorna.

- Internet gav även möjlighet till omdelbar och global koordination av utvecklarresurser, något som tidigare var svårt och extremt dyrbart.

[3]Inom postmodern teoribildning talas ofta om "stora narrativ", d.v.s. berättelser som är så arketypiska att de styr upp vårt tänkande till vissa bestämda, invanda mönster. David-mot-Goliat är onekligen ett sådant stort narrativ.

[4]Ytterst är detta nog en klinch mellan materialistisk och idealistisk historieuppfattning: den förra menar (in absurdum draget) att tillgång och efterfrågan styr historiska skeenden så att du bakom varje skeende finner att det egentligen handlar om tillgång till vissa nyttigheter (Karl Marx är känd för att tillhöra denna skola), och den senare menar (in absurdum draget) att historien drivs fram av starka individers bedrifter, varför det blir ett evigt beskrivande av "stora män". (Herman Lindquist kanske kan tänkas tillhöra denna tradition, även om han knappast skulle skriva under på det.) Historievetenskapen försöker i allmänhet att åstadkomma en syntes av dessa två synsätt. Walker erkänner sig mycket riktigt som motståndare till det materialistiska synsättet i en kommentar till artikeln: "I view this in much the same way that I view neo-libertarians who have a love affair with the self-made man (or woman). Just as I don't believe in the self-made man, I also don't believe in the supremacy of the individual open source author."

- Mängden människor som använder datorer har ökat, och detta ökar naturligtvis även mängden användare av fri programvara och mängden potentiella utvecklare i fria mjukvaruprojekt. (Även detta är till stor del en effekt av Internet.)

- Eftersom mängden programmerare har ökat, så har priset för den tjänst programmerare tillhandahåller, själva programmeringsarbetet, minskat till följd av lagen om tillgång och efterfrågan. Internet ger en mängd kommunikationskanaler som gör det möjligt för den som vill att snabbt lära sig programmering, vilket driver på denna utveckling.

- Den stora mängden programmerare ökar innovationstakten och minskar mjukvaruprodukternas livscykel: dessa måste ständigt uppgraderas.[5] Detta är gynnsamt för fri mjukvara som kan modifieras och distribueras mycket snabbt och till låg kostnad.

- Detta sammantaget har drivit kostnaden för alla de mjukvaror som tidigare såldes som ”shareware” o.s.v. till noll. Ingen förväntar sig längre att kunna sälja ett mindre datorprogram. Priset för alla standardmjukvaror går asymptotiskt mot noll. Den stora användarbasen och tillgången till en sådan stor mängd utvecklare gör att det blir omöjligt att tillhandahålla standardmjukvaror på något annat vis än via fri mjukvara, eftersom användarbasen är så stor att den uppnår egenmakt[6] — d.v.s. den behöver inte längre förlita sig på mjukvaruproducerande företag eller individer.

- De enda utvägar en mjukvaruproducent finner är till sist två: antingen att tillhandahålla funktionalitet som ingen annan producent (inlusive de fria mjukvarorna) kan tillhandahålla, eller att släppa källkoden fri och bli ett fritt mjukvaruprojekt,[7] ty det återstår inga andra värdeskapande mekanismer.

- Även om en utvecklare eller ett företag väljer att följa det första alternativet kommer de att förlora gentemot alternativ två, eftersom

[5] Detta har noterats inom diskursen för det s.k. ”moderna projektet” och brukar sammanfattas i ett citat från Karl Marx där han sammanfattar den moderna tiden med orden: *”allt fast förflyktigas”*.

[6] Det engelska ord som här översätts med *egenmakt* är *empowerment*. Ordet *egenmakt* använd i Sverige till exempel i *egenmaktsutredningen* som handlar just om det som på engelska kallas *empowerment*.

[7] Någon skulle kanske använda uttrycket ”If you can’t beat them, join them.” Bob Gifford anmärker i en kommentar att det finns ett tredje alternativ: att utveckla helt nytänkande mjukvaruprodukter, jfr. t.ex. IP-telefoniprodukten *Skype*. Detta kan möjligen ses som en variant av det första alternativet.

den stora mängden deltagande användare, och därmed utvecklare, gör att det fria mjukvaruprojektet kommer att utvecklas mera över tid.

- Eftersom Internet ständigt expanderar (ännu återstår större delen av jorden befolkning) kommer användarbasen, och därmed utvecklarbasen ständigt att öka. Detta leder i förlängningen till att den fria mjukvaran tar över allt mer.

Summa summarum var det av ekonomiska skäl alldeles oundvikligt att en stor mängd fri mjukvara skulle uppstå i denna tid.

Walker, liksaom andra observatörer talar om att mjukvara ”kommodifieras”[8] vilket innebär att det är en konsumentprodukt: alla behöver det, alla använder det, alla kommer att skaffa sig det i den variant som har högst prestanda till lägst pris. Få eller inga personer behöver ”något extra” från mjukvaran som inte redan finns i alla tillgängliga alternativa produkter. Eftersom fri mjukvara tenderar att ge högst prestanda till ojämförbart lågt pris kommer den alltid att vinna.

Bland de företag och enskilda personer som väljer att använda och bidraga till fri mjukvara handlar det definitivt ofta om ett rent ekonomiskt avvägande och ingen altruistisk handling: de företag som säljer Linuxdistributioner säljer konsultation, långsiktighet och drift av infrastruktur. Många säljer en utvecklingskedja eller plattform för inbyggda system. Vissa företag söker bara en kostnadseffektiv bas att köra sina proprietära stordriftssystem på, såsom t.ex. Oracle eller SAP/R3 eller de otaliga Apache-webbservrar, PHP-skripfarmer, MySQL-databaser och liknande som erbjuds via mindre Internettjänsteföretag. De företag som slutit upp omkring projektet Eclipse har ofta gjort det av det skälet att de inte har råd eller lust (att det avviker från kärnverksamheten) att utveckla något eget grafiskt utvecklargränssnitt. De som kommer till Linux från andra POSIX-varianter söker ofta ett enhetligt, modernt och ordentligt standardiserat system.

Ytterligare en bidragande faktor kan vara själva datorns kommodifiering: på den tiden en ny persondator kostade 50.000 kronor var kostnaden för operativsystemet och tillämpningsprogrammen i princip liten. I dag utgör denna kostnad allt större del av det totala inköpet när du vill skaffa dig en ny dator. Bland annat Eric S. Raymond har spekulerat i att kostnaden för operativsystemet kommer att nå en magisk gräns. När t.ex. operativsystemet kostar lika mycket eller mer än själva hårdvaran, datorn, kommer något drastiskt att ske: leverantörer av ”datorpaket” kommer att välja att förinstallera fria mjukvaror för att få ned kostnaderna och höja marginalerna.

[8] En svengelsk term med ursprung i det engelska ordet *commodification*.

3.2 Antropologisk, sociologisk, eller psykologisk förklaring

Den mest utbredda och lästa artikeln om den fria mjukvarans sociala mekanismer är utan tvekan *Katedralen och basaren*[9] (se [25]) av Eric S. Raymond. Artikeln handlar mycket om *varför* fri mjukvara är så pass bra: bland annat framhålls att kvalitén kommer sig av att många granskar källkoden,[10] och att alla som kan är välkomna att deltaga. Men vi skall här försöka förklara de deltagandes motivationer.

Eric Raymond kallar själv sin analys för antropologi (läran om kulturer och samhällstyper) men är nog snarare sociologisk (läran om samhället) eller rent av psykologisk (läran om hur och varför människor tänker och handlar på vissa sätt). Raymond är ingen särskilt renommerad antropolog, men har å andra sidan gjort omfattande deltagande studier som utvecklare i flera fria programvaruprojekt.[11]

Hans huvudpunkter för *varför* människor tillverkar fri mjukvara[12] är dock:

- Att programmerarna löser problem som de själva upplever, och inte tvingas till att vända sig till tredje part för att få dem lösta.
- Att de upplever en stark belöningskänsla i form av uppskattning från andra programmerare och användare av programmen, och i form av att andra blir imponerade över vad de åstadkommit.
- En stark känsla av att programmet, trots att det är fritt, är programmerarens egendom. Den ursprungliga programmerarens hävdarätt till programmet (i form av dennes rätt att koordinera utvecklingen) erkänns så gott som alltid inom den fria programvarukulturen.

Raymond är som så många andra ekonomisk reduktionist och ställer upp alla sina resonemang som om de rörde nyttigheter. Han talar t.o.m. om en *nyttighetsfunktion* (engelska: *utility function*) som består av ovanstående punkter. Notera dock den tydliga glidningen i begreppet

[9] Och dess uppföljare *Bopålar i Noosfären* samt *Den magiska kitteln*.

[10] I detta sammanhang myntar han det han kallar ”Linus lag”: *med tillräckligt många ögon, blir alla buggar synliga.* (Given enough eyeballs, all bugs are shallow.)

[11] När den så ansedde sociologen Manuel Castells häromåret lät publicera boken *Internetgalaxen*, som delvis handlar om fri programvara och dess historia, väntade jag mig en mer grundlig analys om bakomliggande motivationer bland deltagarna. Castells nöjde sig emellertid med att återge vad Raymond och andra redan sagt och kan inte påstås ha tillfört någonting alls till diskussionen om fri programvara.

[12] Eric S. Raymond använder inte begreppet *fri programvara* utan *open source*. Resonemanget är dock detsamma.

egendom: Raymond använder detta ord även om saker som varken rör rätten till ekonomiskt utnyttjande eller andra juridiska rättigheter, utan på ett vis som enbart refererar till sociala beteendemönster.

Man kan tycka att detta är ett lite smått underligt synsätt, i varje fall gör jag det: den del av programmerarnas belöning som handlar om gensvar från användare och erkänsla bland andra utvecklare kan lika gärna kallas *social interaktionism* och förklaras socialpsykologiskt. Johan Asplund menar t.ex. i sin bok *Det sociala livets elementära former* att vår drift att interagera socialt med andra människor är en fullständigt grundläggande och näst intill mekanisk reflex.

Linus Torvalds beskriver något mycket snarlikt i slutet av sin egen bok *Just for Fun* där han avhandlar orsakerna till att han skrev Linux, han identifierar tre möjliga motiverande faktorer:[13]

1. Överlevnad (Torvalds understryker att det inte handlar om pengar utan mat på bordet)

2. Sociala relationer, från person-till-person till hela samhället.

3. Underhållning. Det är roligt. Detta är också anledningen till att boken heter *Just for Fun*.

Det har sagts att GNU/Linux gör att det återigen blir ”roligt” att använda datorer, och det gäller både utvecklare och användare. Uppstyrningen, det abstrakta ”kund”-tänkandet och den skoningslösa produktifieringen som kännetecknar proprietär programvara förekommer inte inom projekt för fri mjukvara, annat än som interna begrepp hos de företag som deltar i dessa projekt på sina egna villkor. Grunden ligger alltid i en ömsesidig omedelbarhet och direkthet.

Vad menar en programmerare med att det måste vara ”roligt” att använda en dator? Genom att iaktta några av den fria mjukvarans portalfigurer kan vi se ett mönster: de har drivit oavlönade programmeringsprojekt på sin fritid, eftersom det är så roligt att programmera. När de kommit i kontakt med den industriella mjukvaruindustrin och dess villkor har de upptäckt att denna har tagit bort flera av de element som gjorde programmeringsarbetet roligt.

Denna motivation är ytterligt stark: Richard Stallman representerar en äldre generation av sådana programmerare och har många gånger beskrivit varför han inte anställde sig själv för att arbeta för Free Software Foundation: ”jag visste att jag kunde få Richard Stallman att arbeta gratis” är hans motivation.[14]

[13] Dessa återfinns på sidan 250-251 i pocketupplagan av *Just for Fun*[33].

[14] Sagt under ett föredrag vid Lunds Tekniska Högskola. Det går dock ingen nöd på Richard Stallman eftersom han senare tilldelats ett livstidsstipendium.



Figur 3.1: Felrapporteringssystemet *Bugzilla* är välkänt och nödvändigt för alla som aktivt använder fri mjukvara.

Flera företrädare för fria mjukvaruprojekt bär på liknande historier. Många av dem kunde ha arbetat med bättre lön och status, men hade de då haft lika roligt, och hade de varit lika kreativa som de är i sina fria mjukvaruprojekt? Antagligen inte. Det är såklart möjligt att uttrycka detta som att denna frihet är värd si-och-så mycket pengar, men det missar helt poängen.

3.3 Att deltaga i fria mjukvaruprojekt

Att använda fri mjukvara är till viss del förbundet med att vara deltagare. Om någon annan sköter installation och underhåll av ditt operativsystem, så är det lämpligt att du vänder dig till denna person för hjälp, men är du själv inblandad på något vis, är det lämpligt att du på något plan betraktar dig som deltagare i de fria mjukvaruprojekten.

Om du använder en distribution av GNU/Linux tar distributören också ett visst ansvar för att rätta till problem och felaktigheter som uppstår i mjukvaran. Detta gäller naturligtvis *bara* om du har *köpt* GNU/Linux på t.ex. en CD-skiva i ett paket för några hundra kronor. Detta är en del av det som du betalar för.

Om du har laddat ned din distribution eller installerat nya program som inte följde med den, är det enda sättet att avhjälpa fel att rapportera dem direkt tillbaka till projektet. Detta kan ske på flera sätt: dels via epostlistor där du kan komma i kontakt med utvecklare direkt. Dels har utvecklarna personliga epostadresser via vilka du kan kontakta dem direkt.

Större projekt har ofta ett felrapporteringssystem där du kan rapportera in felen och se till att de delges rätt utvecklare. I figur 3.1 ser du Mozilla-projektets *Bugzilla*; ett avancerat och bra felrapporteringssystem.[15] I detta kan du också följa upp hur dina fel behandlas av utvecklare längs vägen, och vad de gör åt saken, eller om de rent av inte gör något åt saken och i så fall varför.

Somliga personer klagar ibland på att detta system är godtyckligt, och att det inte finns några garantier för att ett anmält fel faktiskt blir rättat. Detta är sant, men handen på hjärtat: hur många proprietära program har du anmält fel på och fått dem rättade? Har du kanske rent utav inte ens tänkt tanken att kontakta t.ex. Adobe och be dem lösa ett problem du har med ett program?

Om du arbetar för ett större företag som använder ett visst program, och behöver få ett fel rättat snabbt, eller snabbt behöver får något tillägg utarbetat, är fri mjukvara bättre än proprietär: för det första kan en distributör av fri programvara sälja garanti för fast kostnad eller på löpande räkning för att rätta till fel som uppstår. Om distributören misslyckas att åtgärda ett fel, kan du själv hyra in en programmerare, som tar källkoden till programmet och lagar felet. Lagningen kan sedan kommuniceras tillbaka till projektet så att felet inte inträffar igen.

Att proprietära mjukvaruleverantörer skulle göra motsvarande operation gratis är inte sant: i de fall jag fört diskussioner med stora leverantörer om sådana saker har det alltid landat på att de förväntat sig speciell betalning för att åtgärda sådana fel snabbt.

Minns också noga att fri programvara inte betyder att allt är gratis.

Somliga fria mjukvaruprojekt gör det möjligt att betala en av projektets topprogrammerare för att snabbt lösa ett specifikt problem för en engångssumma, men denna företeelse är ännu inte särskilt vanlig.

3.4 Licenser

> ”Åt var och en efter behov, av var och en efter förmåga.” Klart Linux är kommunism (...) Det finns ett namn för kombinationen av kommunismens krav på lika ekonomiska förutsättningar och frihet för individen. Anarkism! Linux är alltså anarkism. (...) Linux är liberalism, för det är väl ändå det som först och främst förknippas med frihet. (...) Linux är konservatism, det bygger ju på lastgammal Unix. (...) Li-

[15] I branschen kallas detta ibland för ”Change Management Tool”. Låt dig inte luras av sådana floskler. Fråga dig istället vad systemet faktiskt gör. Ett felrapporteringssystem är ett felrapporteringssystem är ett felrapporteringssystem.

> nux är nazism: Låt [bara] den bästa källkoden överleva och dominera världen!
>
> — Olika kommentarer i bloggen Gnuheter på frågan ”Är Linux kommunism?”

De fria mjukvaruprojekten använder olika *licenser* vilka i korthet kan sägas vara juridiska avtalsvillkor kopplade till programmerarens upphovsrätt (copyright). För att programmen skall få användas, distribueras, kopieras o.s.v. krävs att användaren följer licensen. Dessa villkor upprättas av den som skriver ett datorprogram, och som därmed är dess *upphovsman*. Dessa villkor är i stort sett inte annorlunda från de som omfattas av proprietära mjukvaror. Rättigheterna för en upphovsman definieras i upphovsrättslagen, och möjligheten att utforma licenser definieras och begränsas av avtalsrätten.[16]

En juridisk individ som exempelvis ett företag kan inte vara upphovsman till ett datorprogram, men kan däremot vara rättighetsinnehavare och tillämpa en viss licens på sina program. De programmerare som arbetar på företaget har då överlåtit sin upphovsrätt till företaget i någon form av kontrakt som undertecknades i samband med anställning vid företaget.[17]

Innan du använder en fri mjukvara bör du ha någon slags uppfattning om dess licens. Detta avsnitt är tänkt att presentera några sådana licenser. Du är som användare *tvungen* att följa dessa villkor, det går inte att komma efteråt och försöka kompromissa bort villkoren i licensen. Om du exempelvis ogillar GNU GPL av ena eller andra skälet, bör du också i ärlighetens namn undvika att använda program som har denna licens. (Vilket i praktiken är alla GNU/Linux-distributioner som finns.)

Vissa frågor har rests om huruvida dessa programlicenser är förenliga med svensk rätt. Den enda mer systematiska utredning som gjorts av den frågan är rapporten *Upphovsrättsliga aspekter på licenser för fri programvara och öppen källkod*[23] av Jessica Olofsson vid institutet för rättsinformatik på Stockholms Universitet. Denna är lätt att läsa och obligatorisk läsning för den som behöver veta mer om licenser av denna typ. Hennes slutsats är:

> När det gäller datorprogram inskränks avtalsfriheten (...) av de tvingande reglerna om säkerhetskopior, observationsrätt

[16] Avtalsrättslagen heter *Lagen om avtal och andra rättshandlingar på förmögenhetsrättens område*.

[17] Upphovsrätten för program skapade av en anställd programmerare på ett företag övergår per automatik till arbetsgivaren (40 a § URL) men närstående rättigheter och t.ex. regler om sekretess avtalas separat.

> och rätt till dekompilering (...) Licenserna för fri programvara och öppen källkod torde inte medföra någon konflikt med dessa, då användarens förfoganderätt istället utökas på dessa områden. (...) Det torde i och med detta inte finnas något principiellt hinder mot att tillämpa licenser för fri programvara och öppen källkod i svensk rätt.

I det följande kommer några av de vanligaste licenserna att presenteras på ett kortfattat lekmannamässigt vis.

3.4.1 GNU General Public License, GPL

GNU General Public License, mest känd som bara *GPL* är den vanligaste och mest använda licensen för fri programvara. Den används för c:a 70% av alla fria programvaror. Många programmerare använder regelmässigt GPL, eftersom det är en vanlig och accepterad licens som visat sig fungera bra. GPL används bland annat av alla program framställda inom GNU-projektet och av själva Linuxkärnan. De svenska företagen MySQL AB och Axis Communications har båda producerat stora mängder programvara under denna licens.

GNU GPL anses också vara den rent juridiskt sett mest välskrivna av licenserna för fri programvara. Den har formulerats med hjälp av jurister och har dessutom reviderats en gång och är därför mycket genomtänkt.

Den huvudsakliga avsikten med GNU GPL är dels att uppfylla de fyra friheterna som räknades upp inledningsvis, dels att ställa ett ytterligare krav, nämligen att all programvara som framställs genom att bearbeta ett program som täcks av GNU GPL också måste uppfylla GNU GPL. Det är förbjudet att ändra licensen för programvaran eller modifierade versioner av den, och om t.ex. patent eller domstolsutslag fastslår att mjukvaran inte kan distribueras under GNU GPL, måste distributionen upphöra helt.

GNU GPL *tillåter* också en hel del saker, dels explicit och dels implicit. Det är tillåtet att sälja garanti för programvaran — d.v.s. utlova att den skall uppfylla vissa krav mot en avgift. Det är också tillåtet att sälja själva programvaran på t.ex. en CD-skiva. Det är alltså *inte* fråga om något förbud att bedriva kommersiell verksamhet med mjukvaran.

Det är också tillåtet att blanda GPL-program och icke-fria program på en CD-skiva eller liknande, däremot är det förbjudet att låta ett icke-fritt program länka till[18] ett fritt program under GNU GPL.

[18]Se avsnitt 5.4.5 på sidan 195 för en närmare förklaring om vad som menas med länkning.

Det är dessutom tillåtet att bearbeta och ändra ett program för internt bruk, t.ex. på ett företag, utan att distribuera ändringarna. Så fort programmet avsiktligt sprids eller säljs till en allmänhet eller t.ex. ett annat företag, kommer dock villkoren om tillgång till källkod o.s.v. omedelbart att träda i kraft.

GPL innehåller även ett slags ”giftpiller” mot mjukvarupatent: om ett program under GPL använder någon patenterad metod, och om denna metod är patenterad av upphovsmannen till programmet, kommer licensen att innebära att upphovsmannen även licensierar detta patent för användande i detta program samt *även* att ge *alla andra* GPL-skyddade program fri tillgång till detta patent. Företag som inte släppt sina program under GPL är det däremot fritt fram att stämma.

Bara den som erhållit programmet i någon form kan ställa krav på tillgång till källkoden. Detta innebär, att om du t.ex. köper en DVD-spelare som innehåller någon form av fri programvara licensierad under GPL, så har du rätt att få tillgång till källkoden. Däremot har du inte rätt att kräva det om du inte äger en sådan DVD-spelare. Det finns inget ”krav” på att källkoden skall finnas på Internet, även om den så gott som alltid gör det. Företaget som gjort DVD-spelaren kan nöja sig med att skicka källkoden till dig på en CD-ROM-skiva, men kan samtidigt inte förbjuda dig att i andra ledet publicera källkoden på Internet.

GNU GPL förbjuder *inte* upphovsmannen att samtidigt tillhandahålla programvaran under en annan licens, s.k. *dubbellicensiering*.

Så länge ingen utomstående varit inblandad i projektet, kan den som skrivit ett program välja att släppa det som *både* fri programvara under GPL *och* som proprietär programvara. Denna princip används t.ex. flitigt av svenska MySQL AB, som säljer sin databas MySQL under dels en klassisk, restriktiv licens för de som behöver bygga proprietära system baserade på MySQL, dels under GPL för alla som kan tänka sig att uppfylla villkoren för fri programvara. Denna kombination har visat sig mycket framgångsrik.

GNU GPL brukar användas av programmerare som anser att det program de utvecklar *måste* och *skall* förbli fritt, och därför inte tvekar för att *tvinga* de som bryter mot licensen att uppfylla kraven. De som inte gör det riskerar att stämmas av upphovsmannen till programmet i en civilrättslig process. De som har en mera ”pacifistisk” inställning till denna eventuella konflikt brukar föredra BSD- eller MIT-licenserna. (Se nedan.)

Även om brott mot licensvillkoren bara kan bli föremål för en civilrättslig process saknas inte resurser för de personer och företag som använder GNU GPL. Ett uppmärksammat fall rörde svenska MySQL AB,

som anklagade NuSphere[19] för brott mot GPL. Detta ledde till att experter från Free Software Foundation inkallades, och tvisten avgjordes till MySQL:s fördel. Om fallet är principiellt intressant kan FSF komma att inkallas. Det är alltså inte på minsta vis så att licensen inte skall tas på fullt allvar, även om den kan upplevas som oortodox.

GNU Lesser General Public License, LGPL

GNU *Lesser General Public License*, allmänt kallad *LGPL* är en modifierad version av GNU GPL som tar bort villkoret om att icke-GPL-program inte får länka GPL-program.

Denna licens skapades för att sprida användandet av programbibliotek som framställts inom GNU-projektet, framför allt GNU C Library[20] som var avsett att användas tillsammans med GNU C Compiler, en kompilator för programspråket C. För att uppmuntra användandet av kompilatorn fordrades att programmerare skulle kunna använda GNU C Library utan att behöva släppa alla sina program under GPL. Därför utarbetades LGPL som en kompromiss.

Avsikten med LGPL är alltså, att själva programbiblioteket måste finnas tillgängligt på samma villkor som gäller för GPL. Program som i sin tur använder detta programbibliotek behöver inte följa GPL eller LGPL.

3.4.2 BSD-licensen

BSD-licensen är mest känd för att den just används av de olika BSD-varianterna. Denna licens är mycket tillåtande, den tillåter i princip allt: det enda som ingår är en friskrivningsklausul för eventuella fel i programvaran.

Koden från ett program som licensierats under BSD-licensen kan användas i proprietära program. Så har också skett i mycket stor utsträckning: Microsoft använde exempelvis BSD:s TCP/IP-stack[21] i tidigare versioner av Windows, eftersom det innebar en genväg att snabbt tillföra systemet Internetfunktionalitet.

Programmerarens huvudsakliga syfte med BSD-licensen brukar vara att teknologin och filosofin med programmet skall spridas, i såväl proprietära som fria mjukvaror. (En annan vanlig anledning redogörs nedan under MIT-licensen.)

[19] NuSphere ägs i sin tur av Progress Software.

[20] Se avsnitt 5.1 på sidan 173.

[21] Se vidare avsnitt 9.2 på sidan 300 om vad detta kan tänkas betyda.

BSD-licensen innehöll tidigare ett krav om att den som skapat det ursprungliga programmet skulle få erkännande i programmets dokumentation, eller via t.ex. en *about*-ruta i ett grafiskt program som använde BSD-kod. Detta krav har numera upphävts vad beträffar själva BSD.

Utöver detta innehåller den en rad friskrivningsklausuler från fel på programvaran etc.

BSD är, sedan kravet på att upphovsmannen skall nämnas i programmet borttagits, vad som brukar kallas *kompatibel* med GNU GPL. Det innebär, att en programvara som licensierats enligt BSD-licensen kan licensieras under BSD-licensen *plus* GNU GPL, så att de stränga villkoren från GNU GPL läggs "ovanpå" de ursprungliga kraven, utan att någon motsägelse uppstår mellan de två licenserna.

3.4.3 MIT-licensen

> Jag anser att folk skall göra sina egna val. Jag anser att friheten ligger i friheten att göra vad du vill med det du producerar. (...) Det är trevligt när de [som använder mjukvaran] delar med sig i sin tur, men jag kräver inget sådant på förhand.[22]
>
> — David Dawes

MIT-licensen[23] är unik på så vis att det är den mest liberala (om uttrycket tillåts) av alla licenser för fri mjukvara. En variant av denna licens används av det mycket viktiga projektet XFree86.

Många av de som använder MIT-licensen (och även många som använder BSD-licensen) är till synes inspirerade av Ghandi: de vill inte påtvinga någon några villkor alls rörande mjukvaran, inte att den skall vara fri, inte att deras namn skall nämnas, egentligen ingenting alls. De tycks säga: "Här är en vacker blomma, tag den om du vill, tacka mig för den om du vill, förstör den eller döda den om du vill. Jag tvingar dig inte till något."

Detta resonemang tilltalar många utvecklare av fri mjukvara som därför valt just MIT-licensen. De som är anhängare av GNU GPL brukar mot detta hävda att det är naivt, och att ingen producent av proprietär mjukvara kommer att ändra sin inställning till hur mjukvara skall

[22]Min översättning, källa se: [3].

[23]Namnet kommer från Massachusetts Institute of Technology (MIT) i Boston, där licenser av detta slag användes i projekt som Athena och dess fönstersystem *X* i syfte att sprida idéerna i näringslivet.

framställas genom att programmerare frivilligt ger dem frukten av sitt arbete utan att ställa motkrav.

Det kan vara värt att nämna att Microsoft uttalat sig mycket positivt om BSD- och MIT-liknande licenser, och mycket negativt om GNU GPL. Detta präglar också mycket av tänkandet inom den traditionella programvarubranschen: ge oss gärna era mjukvaror, men inte på några villkor alls. De flesta programmerare som använder MIT-liknande licenser har dock inte denna inställning, utan vill just bara vara kravlösa i största allmänhet, då de anser att en sådan inställning är bäst för mjukvaruvärlden, eller rent av världen i stort.[24]

3.5 Versionsnummer

De fria mjukvaruprojekten delar ett säreget versionsnumreringssystem inspirerat av den proprietära världens mjukvaror. Ett versionsnummer för en fri mjukvara kan se ut såhär:

XFree86 4.3.1

Då är det första namnet på mjukvaran, i detta fall den fria implementationen av X Window System (fönstersystemet *X*), större version 4, mindre version 3, revision 1. Numreringen börjar oftast *före* 1.0, exempelvis fanns Linux 0.12 tillgänglig vid något tillfälle under tidigt 1990-tal, och webbläsaren Mozilla hade länge versioner under 1.0. Den sista siffran, revisionssiffran, brukar uteslutas om den är 0 men kan ibland inkluderas i alla fall, som i *XFree86 4.3.0*.

När du laddar ned en fri mjukvara för att kompilera den själv (vilket inte rekommenderas för en nybörjare) kommer den ofta i en komprimerad fil med namn som *XFree86-4.3.0.tar.gz* eller *linux-2.4.22.tar.bz2*. I båda fallen rör det sig om filarkiv gjorda med kommandot **tar** och sedan komprimerade med **gzip** respektive **bzip2**. Det är en bra övning att någon gång ladda ned ett sådant arkiv bara för att titta på det, och få grepp om hur källkoden för ett fritt mjukvaruprojekt egentligen ser ut.

Utöver detta kan paketeringar som görs av olika distributioner (vilka kommer att avhandlas senare) ha ytterligare ett revisionsnummer, som gäller själva paketet. Dessa har då t.ex. formen *XFree86-4.3.1-1.rpm* eller *xfree86-common_4.3.1-13_all.deb*. Detta betyder att det är RPM-paket

[24] Från och med version 4.4.0 av XFree86 ändrades deras version av MIT-licensen till en GPL-inkompatibel version som liknade den gamla BSD-licensen: det blev nödvändigt att namnge upphovsmännen till programmet. Det är en ganska vanlig åsikt bland utvecklare av fri programvara att upphovsmannen skall anges, och så sker också oftast. En kontrovers uppstod då XFree86 ville göra detta villkor tvingande. Situationen är ännu inte löst.

nummer 1, respektive DEB-paket nummer 13 av huvudkomponenten i XFree86 4.3.1.

När det gäller exempelvis Linuxkärnan och skrivbordsmiljön GNOME, har dessa mindre versionsnummer som skall tolkas som att:

- *udda* nummer representerar utvecklarversioner som inte är färdiga och inte skall användas av vanliga användare, medan
- *jämna* nummer är avsedda som "skarpa" versioner, som skall kunna användas generellt.

Exempelvis är versionerna 2.0, 2.2, 2.4 och 2.6 "skarpa" versioner av Linuxkärnan, medan GNOME 2.5 är en ren utvecklarversion som inte kan garanteras fungera över huvud taget. Utöver detta kommer sedan revisionsnumren.

Som om detta inte var nog kan vissa utvecklare tycka att de treställiga numren trots allt inte är nog. Då dyker det upp paket och filarkiv med namn som *linux 2.4.3-pre23*. Alla dessa underliga namnkombinationer kan omöjligen listas, men här är några:

rc — *release candidate*, ska vi släppa version 4.3.1? Vi gör först en kandidat, 4.3.1-rc1 och om den funkar släpper vi den som 4.3.1, annars går vi vidare med 4.3.1-rc2 o.s.v. tills vi sent omsider kan släppa 4.3.1.

pre — *prepatch*, samma som ovan, men lite mindre säkert om detta kommer att likna slutresultatet eller ej. Namnet kommer av att det fordras en hel del "programlappar" (av engelskans *patch*; jämför att lappa ihop ett hål i ett par byxor) innan denna version kan släppas, och dessa versioner levereras oftast också bara som sådana "patchar".

test — vi testar innan vi släpper. Linux 2.6.0-test3 är den tredje testomgången för version 2.6.0 av Linuxkärnan. När vi till slut känner att den är färdig får den heta 2.6.0.

KAPITEL 4

Distributionerna

Ordet *distribution* är kritiskt för att förstå GNU/Linux-världen. Det är sant att det finns ett projekt som heter GNU och det är sant att det finns en kärna som heter Linux, ett fönstersystem som kallas *X*, och ett par olika desktop-system, men det användaren upplever är i slutänden nästan alltid en distribution.

Med en distribution menas vanligen ett körbart system, d.v.s. en uppsättning program levererade på något digitalt medium, som när de installeras på en dator formar ett fullständigt fungerande operativsystem *och* inkluderar ett antal standardapplikationer som utför sådant som en normal användare vill kunna göra. Om du för första gången skall installera GNU/Linux är det fråga om att du måste välja en distribution, i annat fall är det svårt eller omöjligt att komma igång med operativsystemet.

Förväxla inte distributionen med ”Linux” (vilket enbart är kärnan), lika lite som du förväxlar den med något annat.

Distributionernas historia började i februari 1992 med MCC Interim Linux[1], och följdes snart av bland annat TAMU Linux[2] och Softlanding Linux Systems Linux, SLS Linux, som sattes samman av Peter MacDonald.

Syftet med dessa första distributioner var klart och tydligt: de var avsedda för människor som inte ville bygga ihop sitt eget operativsystem från grunden, utan helst ville installera det färdigbyggt, så som

[1] MCC utläses Manchester Computing Centre, England

[2] TAMU efter Texas Agricultural and Mechanical University

flera andra operativsystem installeras. Innan distributionerna dök upp fanns inga sådana möjligheter — den som ville köra Linux fick bygga det själv — men det är då värt att betänka att Linux 0.12 (som var den version som användes i de första distributionerna) inte bestod av särskilt många delar. Den mängd program som fanns till denna version av Linux var inte heller särskilt stor. Att bygga ett helt operativsystem själv med Linux som bas var faktiskt inte särskilt svårt, och hjälp fanns att få.

Den tidiga distributionen SLS Linux underhölls inte, och av den anledningen tog Patrick Volkerding SLS och skapade en egen variant som han därefter underhöll. Denna släpptes den 17 juli 1993 och finns än idag under namnet Slackware Linux. Volkerding underhåller fortfarande Slackware, numera i samarbete med andra entusiaster.

En annan person som inte kunde stå ut med SLS var Ian Murdock. Denne startade därför ytterligare ett projekt baserat på SLS, som fick namnet *Debian*. Detta namn var en sammanslagning av hans frus namn Debra, och hans eget, Ian. Detta är den näst äldsta distributionen, och började utarbetas den 16 augusti 1993.

Därefter började allt fler distributioner introduceras i en strid ström.

I detta kapitel kommer några vanliga distributioner och deras respektive meriter att presenteras. Detta brukar vara en het potatis, och i princip skulle det kunna finnas lika många olika distributioner som det finns tekniskt kunniga GNU/Linux-användare, men majoriteten har ändå valt någon av de som presenteras här. Det vore orätt att tala om dessa distributioners för- och nackdelar, eftersom alla distributioner har en egen grundtanke och är i princip perfekta för det som de är avsedda för.

Flera datortidningar gör ofta stort nummer av att lista hundratals olika underliga linuxdistributioner och deras respektive karakteristika, men sådana sammanställningar ger sällan någon vettig information. Det är bättre att gå djupt in på några vanliga distributioner och de principer de bygger på, så att du förstår några övergripande huvudidéer bakom dem alla. För en fullständig listning av alla existerande distributioner hänvisas till webbsidan *Distrowatch*.[3]

Du bör ha en ganska klar uppfattning om vad du vill att din distribution skall kunna och vilka principer den skall bygga på innan du skaffar dig ett installationsmedia; läs därför gärna igenom de avsnitt som behandlar respektive distribution åtminstone översiktligt.

[3] Se: `http://www.distrowatch.com/`

4.1 Hur distributionerna fungerar

Följande karakteriserar en GNU/Linux-distribution:

- Distributionen är *inriktad på slutanvändare* som inte vill bygga sitt eget operativsystem från grunden. De kan vara profilerade mot olika användare med olika grad av tekniskt kunnande, men är alltid inriktad på att göra det enkelt att ”komma igång”. (Undantag: Linux From Scratch, se detta avsnitt.)

- Distributionen har ett *installationsprogram* som kan startas från installationsmediet.

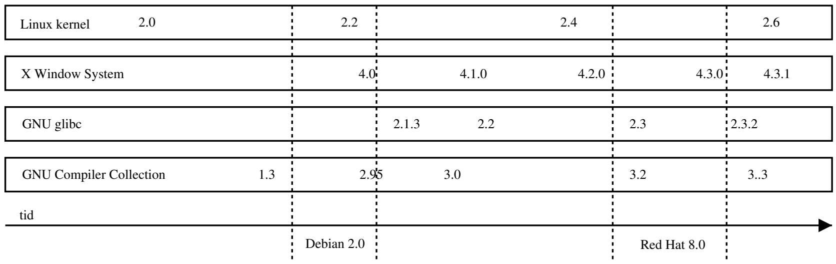
- Med jämna mellanrum kommer det *nya versioner* av distributionen. Detta innebär att distributionerna försöker följa de fria mjukvaruprojekten och föra samman dem på ett sätt som gör att de kan användas tillsammans utan problem.

- Distributionen består av ett stort antal olika mjukvaror, som förhoppningsvis *fungerar tillsammans*. Mjukvarorna kommer mestadels från flera olika fria mjukvaruprojekt, varav själva kärnan Linux bara är ett. GNU-projektets mjukvaror är minst lika viktiga för att en distribution skall fungera överhuvudtaget.

  De olika mjukvarorna ligger oftast inkapslade i andra, mindre *programpaket*. Dessa paket kan vara enkla *tar*-filer, eller komplexare paketstrukturer som *RPM* eller *DEB*. Paketen innehåller ett antal filer och deras positioner i filystemet, kataloger som måste skapas, skrip som skall köras i samband med installation och/eller avinstallation samt information om vilka paket ett visst paket i sin tur är beroende av. Längst ner i beroendekedjan brukar i allmänhet finnas ett paket som definierar rotfilsystemet och dess kataloger.

  Programpaketen bygger upp det totala operativsystemet på samma vis som tegelstenar bygger upp ett hus: du kan välja vilka delar du vill ha med i ditt bygge, men alla operativsystem måste ha åtminstone vissa obligatoriska paket, som till exempel kärnan och skalet installerade. Paketen kan normalt också uppgraderas individuellt.

  Detta faktum ger också användaren en möjlighet att plocka isär operativsystemet, lära sig en del om dess komponenter som vad de gör, ifall det är lämpligt att uppgradera dem o.s.v. Detta skiljer klart ut GNU/Linux-distributioner från den solida kartong och



Figur 4.1: Illustration som visar hur distributioner sätts ihop som snitt av versioner av de olika fria mjukvaruprojekten. De projekt som ingår i en distribution är i själva verket betydligt fler. Här visas några mycket viktiga projekt jämte distributionerna Debian 2.0 (även kallad Potato) samt Red Hat 8.0. Varje projekt resulterar i ett eller flera paket med mjukvarukomponenter. Bilden gör inga anspråk på exakt historicitet, utan är menad som en illustration av ett möjligt förhållande mellan projekten i det ögonblick en distribution släpps. Beträffande versionsnummer, se sidan 126.

monolitiska karaktär[4] som utmärker andra operativsystem. Komponenttänkandet från POSIX-världen syns här mycket tydligt.

- För att hjälpa till med underhållet av systemet ingår det ofta en del små smarta *hjälpprogram* för att administrera systemet. Dessa är då ofta utvecklade av distributören, d.v.s. den organisation som ligger bakom distributionen. Med tiden har dock en större del sådana smarta hjälpprogram tenderat att bli generella program för allt fler distributioner och därmed program i sin egen rätt, eller övergått till att bli en del av skrivbordshanterare som GNOME och KDE.

- Distributionen har för det mesta en funktion för att *uppgradera* till en nyare version när en sådan kommer. Det är också för det mesta möjligt att löpande uppgradera de mindre paketen, exempelvis för att åtgärda programlöss (buggar), och täppa igen säkerhetshål i den distribution som används.

Distributionerna utgör också en naturlig del av den fria mjukvarans ekosystem, om uttrycket tillåts: de personer som sammanställer distributionerna deltar ofta aktivt i de olika mjukvaruprojekten. I synnerhet

[4]Med *monolitisk karaktär* menas här att operativsystemet upplevs som gjutet i ett enda stycke.

testar de hur bra programmen fungerar tillsammans med andra program, och de lämnar oftast både rättningar och utförliga felrapporter till projektens utvecklare. Om det inte går att få ett program att fungera i distributionen så kommer distributören att underhålla en uppsättning ändringar vid sidan av projektet om så krävs, men oftast är projektet villigt att integrera dessa ändringar.

Distibutioner har olika fokus. En möjlig indelning är:

Fokus på enkelhet — distributionens mål är att den skall vara *enkel* att installera, och *enkel* att använda. Typiska exempel: Red Hat/ Fedora Core Linux, Mandrake Linux, Lindows, SuSE Linux.

Fokus på säkerhet och stabilitet — distributionens mål är att den skall vara säker och stabil. Debian GNU/Linux är speciellt fokuserad på stabilitet. I begreppet stabilitet ingår också stordrift på stora företag: här har såväl Red Hat som SuSE speciella produkter bara avsedda för att användas i stor, stabil skala.

Fokus på konfigurerbarhet — konfigurerbarhet eller styrbarhet innebär att varje detalj i operativsystemet kan detaljstyras, väljas och fintrimmas för hand. Slackware, Linux From Scratch och Gentoo är distributioner som kan sägas ha detta fokus. Dessa distributioner ligger i någon mån ”närmare” projekten, de kräver också ett ganska djupt tekniskt kunnande.

Fokus på en uppgift — distributionen är inte utformat för att vara generell utan skall i princip bara göra en enda sak. Hit hör t.ex. *NAS-Lite*[5] som är avsedd att endast fungera som en filserver, *floppyfw* för datorer som bara skall användas som brandvägg (se vidare sidan 366) eller FREESCO[6] som endast är avsedd att fungera som en paketrouter. Dessa är ofta så små att de kan starta från en floppyskiva eller ett USB-minne. Man kan säga att en sådan distribution gör om en vanlig dator till ett inbyggt system.

Vi ser detta övergripande mönster till viss del gå igen i BSD-världen, där OpenBSD är fokuserad på säkerhet, NetBSD på stabilitet o.s.v. Det finns andra nischade GNU/Linux-varianter: DeMuDi, *Debian Multimedia Distribution* är en version av Debian GNU/Linux speciellt inriktad mot multimedia, FREESCO är utformad för att efterlikna routrar från Cisco, m.fl.

[5]Se: `http://naslite.sourceforge.net/`
[6]Se: `http://freesco.sourceforge.net/`

4.1.1 Installationsmedia

För att installera en distribution krävs någon form av installationsmedia som innehåller hela eller delar av den distribution du tänkt dig. Detta är de vanligaste sätten att få tag i ett installationsmedia:

Köp skivor — det enklaste och mest intuitiva viset att ordna detta är naturligtvis att köpa eller låna en DVD- eller CD-skiva med den distribution du vill ha. Ofta kan sådana skivor medfölja på omslag till datortidningar: de amerikanska Linuxmagasinen brukar ibland till och med ha två.

Att köpa en kartong med manual och DVD-skiva kan kosta cirka 500 kronor, och för vissa distributioner är detta ibland enda sättet att få tag på en installationsskiva. I detta pris brukar då ingå en viss support: ett telefonnummer dit du kan ringa och få råd och hjälp under installation och driftsättning. Om du bara kopierar skivan, får den gratis med en tidning eller laddar ned den från internet får du ingen hjälp alls, du får hjälpa dig själv, läsa dig till det du behöver veta eller fråga andra i samma situation. Tilläggas skall väl att få eller inga distributioner kan erbjuda någon support på svenska.

Ladda ner skivor — det näst vanligaste viset är att ladda ner en distribution från Internet. Här får du dock se upp lite grand: en distribution är *stor* och om du inte har en mycket högkapacitiv nätförbindelse med god tillgång till omvärlden och utan problem kan ladda ner cirka 2 Gigabyte, bör du inte ge dig på detta. Distributionerna brukar ha information på sina hemsidor om vart de kan laddas ned (om det är möjligt). Ett populärt ställe bland svenska användare är de olika nordiska universitetens FTP-arkiv. Följande adresser brukar vara någotsånär tillförlitliga och snabba från det svenska fastlandet:

```
ftp://ftp.sunet.se/pub/Linux/distributions/
ftp://ftp.funet.fi/pub/Linux/mirrors/
ftp://ftp.ntnu.no/pub/linux/
```

I katalogerna här finner du ett antal olika distributioner i form av s.k. ISO-filer, vilket är avbilder av hela CD- eller DVD-skivor som kan laddas ner och brännas på såväl GNU/Linux-, Macintosh- och Windowsdatorer. (Se avsnitt 7.4 på sidan 274 för information om hur det går till i Linux.) De flesta CD-brännarprogram till de flesta operativsystem kan hantera sådana filer.

En distribution kan bestå av flera iso-filer. I allmänhet behöver du ladda ner och bränna alla för att kunna installera distributionen, men Debian är ett viktigt undantag: den består i dagsläget av inte mindre än nio skivor, men innehållet på dessa är rankat efter popularitet, och därför behöver du sällan mer än de två första.

DVD-skivor med distributioner har också börjat dyka upp, liksom DVD-avbildsfiler, som också är ISO-filer, fast större. Att ladda ner en hel DVD-avbild från Internet tar ännu längre tid än att ladda ner en CD-ISO-fil.

Stora iso-filer är svåra att ladda ner från Internet på grund av att anslutningen ofta bryts och FTP-servrarna lätt blir överbelastade, i synnerhet i samband med att en ny version av någon distribution kommer ut. Detta har lett till vissa fantasifulla lösningar, Debian använder t.ex. *Jigdo*,[7] ett program som laddar delar av filerna från olika ställen, och som fortsätter där det slutade om anslutningen bryts. Andra har använt *BitTorrent* (ett slags peer-to-peer-program) för att distribuera filerna. Sätten är många.

Disketter — de flesta distributioner kan i värsta fall installeras från disketter. För gamla datorer är detta ibland nödvändigt. Det finns ofta en speciell katalog med diskettavbilder, s.k. *.img*-filer,[8] som kan sparas ned och överföras till disketter med hjälp av program som t.ex. **rawrite** som används under Microsoft Windows. Om du redan har ett GNU/Linux-system kan du snabbt kopiera en *img*-fil till en diskett med kommandot **dd if=foo.img of=/dev/fd0** eller med nyare GNU/Linux-system kort och gott **cp foo.img /dev/fd0**. (OBS! Skriv inget innovativt efter **of=...**, om du t.ex. skriver **/dev/hda** kommer du naturligtvis att förstöra din hårddisk...) Ofta finns en hjälpfil som beskriver hur detta går till, samt en kopia av programmet **rawrite** tillsammans med diskettavbilderna.

Även om det inte är nödvändigt att installera hela systemet från disketter (vilket kan bli ganska många) kan det ibland vara nödvändigt att starta från en diskett för att få igång installationen från ett annat media. I dessa fall krävs oftast bara 1-2 disketter.

Nätverk — många distributioner stödjer även installation över nätverk, på så vis att du startar från en minimal uppsättning disketter eller en väldigt sparsam CD-skiva, och sedan laddar ned paketen efter hand som de behövs. Detta är främst avsett för de som har ett

[7]Se: `http://home.in.tum.de/~atterer/jigdo/`
[8]Av engelskans *image*, avbild.

eget internt nätverk med en kopia av operativsystemet tillgängligt via FTP, men den som inte är orolig för att plötsligt förlora nätverksförbindelsen eller nekas tillträde till FTP:n kan ju gärna prova detta.

4.1.2 Inventera din hårdvara

Innan du börjar installera en distribution på din dator bör du ha någon slags uppfattning om vilken hårdvara du har i datorn. Moderna PC-datorer är ofta ett hopplock av delar från diverse leverantörer och det finns inte en chans att räkna upp dem alla här. Distributionerna vi kommer att beröra är till för PC: om du har en Macintosh, Sun SPARC eller liknande så fordras specialkunskap utöver vad som avhandlas här.

Tänker du installera GNU/Linux på ett inbyggt system är detta betydligt krångligare, men då är du sannolikt också en betydligt mer kvalificerad användare. Dessa system byggs ofta runt ett s.k. labbkort från någon processor- eller systemtillverkare, och då finns det för det mesta också en minimal distribution som tillverkats av den leverantör som gjort kortet. (I annat fall är du i den roliga situationen att du får vara först med något, vilket inte är lätt.)

Några bra sätt att inventera hårdvara:

- Små broschyrer, disketter, CD-ROM-skivor och liknande som följde med datorn beskriver ofta den ingående hårdvaran i detalj. Du vill ha så mycket teknisk information som möjligt. Tillverkarnas webbsidor brukar också kunna innehålla servicemanualer och liknande tekniska dokument som kan innehålla ytterligare information.

- BIOS-menyn i din dator inehåller ofta mycket information om hårdvaran, samt möjlighet att slå på och av den. BIOS-menyn aktiveras ofta genom någon knapptryckning vid uppstart, t.ex. funktionstangent *F1*, *F2*, *F10* eller *Esc*. Det är inte alltid denna tangentkombination är uppenbar, vilket i sin tur kan leda till en del experimenterande och informationssökande.

- Du kan starta ett minimalt GNU/Linux-system från diskett eller CD-ROM och köra kommandot **lspci**. Detta kommando listar alla enheter som kan hittas av kärnan på PCI-bussen, vilket är den mest använda bussen i dagens datorer.[9] Även kommandot **lsusb**

[9] Informationen om vilka kort som innehåller vilka chipsets kommer från en databas som hör ihop med programmet, och om någon data i denna är felaktig eller saknas kan du faktiskt lägga till det själv (om du lyckas ta reda på det på annat vis). Varje enhet på

kan vara användbart, eftersom det förekommer USB-enheter som byggs in i datorer och inte sitter i de externa portarna. (Se vidare sidan 166 om Knoppix/Gnoppix.)

- Om du tidigare haft ett annat operativsystem på din dator kan detta ge lite information om vad som finns i datorn. I Microsoft Windows finns i ”kontrollpanelen” ofta en ikon för något som kallas ”system”. Läs igenom och anteckna saker som står där.[10]

- Du kan titta på den fysiska utrustningen genom att ta loss korten ur datorn och undersöka dem, titta på baksidan av bildskärmar o.s.v. Med utgångspunkt från detta kan du ofta söka efter mer information om hur du kör GNU/Linux på din hårdvara på Internet. Har du en specifik hårdvara som exempelvis en viss Compaq-modell, är sannolikheten stor att andra har installerat GNU/Linux på den före dig och har många bra tips.[11]

- Om du lyckats installera ett system men har svårt att få vissa delar att fungera finns andra trick som blir möjliga tack vare medföljande program: **XFree86 -configure** som beskrivs på sidan 229 och vidare på sidan 243 kan användas för att detektera såväl grafikkort som bildskärmstyp.

Även om detta inte är en hårdvarubok kan det vara på sin plats att nämna några saker som du bör ta reda på innan du installerar din distribution:

- Om ditt moderkort har speciella funktioner och vilka dessa då är. Sådana funktioner kan vara: speciella energisparfunktioner såsom ACPI,[12] inbyggda modem, nätverks-, radiolan-, bluetooth- och ljudkort (speciellt i bärbara datorer) samt temperatursensorer.

- Vilket grafikkort som sitter i datorn, samt eventuellt hur mycket minne som sitter internt på detta kort, och vilka upplösningar och färglägen det kan leverara.

PCI-bussen har ett unik ID-nummer (två sextonbitarstal, ett som identifierar tillverkare och ett som identifierar produktnummer) och ett unikt subsystem-nummer (två andra sextonbitarstal som används då en viss hårdvara byggs in i en annan hårdvara, t.ex. i en laptop). Om du hittar en ny hårdvara kan du registrera den på webbsidan `http://pciids.sourceforge.net/`.

[10]Om några speciella nätverksinställningar krävs för att ansluta din dator till ett lokalt nätverk eller en uppringd förbindelse, är det förvisso en bra idé att stjäla även dessa från ett annat operativsystem, ifall du inte har tillgång till den informationen på annat vis.

[11]Om du installerar en bärbar dator är det alltid bra att söka hjälp på `http://www.linux-laptop.net/`.

[12]ACPI, *Advanced Configuration and Power Interface* används speciellt i vissa bärbara datorer, men även allt mer i stationära skrivbordsdatorer.

- Vilken bildskärm du har ansluten till datorns grafikkort, mer specifikt vilka uppdateringsfrekvenser den klarar av och vilken rekommenderad upplösning den är tillverkad för.
- Vilket nätverkskort som sitter i datorn, i form av tillverkare och modell.
- Vilket ljudkort som sitter i datorn, i form av tillverkare och modell.
- Om du har styrkort för SCSI[13]-enheter i datorn.
- Vilken typ av tangentbord du har anslutet till datorn. (I allmänhet är detta på våra breddgrader ett 101- eller 102-tangenters svenskt tangentbord.)

Saker som du däremot *inte* behöver veta, är vilken processor som sitter i datorn, vilka portar som finns direkt på moderkortet, huruvida det finns IDE[14]-kontakter på moderkortet och vilka hårddiskar och/eller CD-ROM eller DVD-enheter som sitter anslutna på dessa IDE-kontakter, om det sitter en diskettstation ansluten till moderkortet och andra saker kopplade till moderkortet. Dessa saker är så basala att Linuxkärnan i 99 fall av 100 kan hantera dem standardmässigt.

4.1.3 Bootstrap loader och multiboot

Din distribution med alla dess paket skall ju installeras någonstans, och denna plats är i det normala fallet din dators primära hårddisk. I detta och nästa avsnitt skall vi därför bekanta oss med hur ett operativsystem lagras på en hårddisk på IBM-kompatibla PC-datorer (kort IBM PC), och hur det startas från denna hårddisk. Om du inte tänker installera på en PC, eller om du redan kan allt om hur hårddiskar fungerar kan du hoppa över dessa två avsnitt.

Först betraktar vi frågan om hur vår dator egentligen lyfter sig själv i håret med en bootstrap loader på det vis vi pratade om redan i inledningen på sidan 11.

IBM PC-arkitekturen definierar ett 512 byte stort *huvudbootblock* (engelska: *master boot record* eller bara MBR), även kallat *bootsektor* som pla-

[13]SCSI står för *Small Computer System Interface* och är en standard för att överföra data mellan hårdvaruenheter över en buss, typiskt användningsområde: hårddiskar, CD-ROM-enheter, scannrar.

[14]IDE står för *Integrated Drive Electronics*, och är ett alternativt namn för ATA (*Advanced Technology Attachment*) vilket är det i särklass vanligaste sättet att ansluta hårddiskar, CD-ROM-enheter och diskettstationer till en dator idag.

Huvudbootblocket

Offset	Innehåll
0x000	Bootblockskod i x86-maskinkod.
0x1BE	16 byte-post som beskriver den första partitionen
0x1CE	16 byte-post som beskriver den andra partitionen
0x1DE	16 byte-post som beskriver den tredje partitionen
0x1EE	16 byte-post som beskriver den fjärde partitionen
0x1FE	2 byte ID som alltid är 0xAA 0x55, och avslöjar att detta är en partitionssektor.

Partitionsposterna

Offset	Betydelse
0x00	Partitionsstatus: 0x80 betyder *aktiv* (primär) 0x00 betyder inaktiv (logisk).
0x01	Vilket läshuvud som används på partitionens första sektor.
0x02	Första sektorn som används av partitionen.
0x03	Första cylindern som används av partitionen.
0x04	Partitionstyp. (t.ex. 0x05 = utökad partition.)
0x05	Vilket läshuvud som används på partitionens sista sektor.
0x06	Sista sektorn som används av partitionen.
0x07	Sista cylindern som används av partitionen.
0x08	Placering av bootsektorn i partitionen (4 byte).
0x0c	Totala antalet sektorer som används av partitionen (4 byte).

Figur 4.2: Uppdelningen av de 512 byten (0x200 hexadecimalt) i huvudbootblocket, och hur de 16 byte (0x10 hexadecimalt) stora partitionsposterna är indelade.

ceras allra först på den första (primära) IDE- eller SCSI-hårddisken.[15] Dessa 512 byte innehåller ett datorprogram som startas automatiskt[16] av datorns BIOS och därutöver oftast en partitionstabell. Se figur 4.2 för en illustration av det faktiska utseendet.

I DOS och Microsoft Windows fungerade programmet som låg lagrat i detta huvudbootblock så att det i sin partitionstabell slog upp den *aktiva* partitionen, vilket normalt bara skall vara en. Därefter laddades *bootblocket* från den aktiva partitionen, vilket är 512 utpekade byte, angivet med en offset från partitionens start. Detta innehöll i sin tur ännu ett program och startades sedan på samma vis som huvudbootblocket, för att sedan i sin tur starta upp operativsystemet som var installerat på just denna partition.

Detta gäller inte de två bootstrap loader-program som används i

[15]Den primära hårddisken namnges i GNU/Linux-världen för **/dev/hda** om det är en IDE-enhet eller **/dev/sda** om det är en SCSI-enhet och i Microsoft Windows kort och gott **C:**. Enheten kan även vara en RAID-enhet o.dyl. men sådana krångligheter går vi inte in på här.

[16]Programmet laddas in på adress 0x7c00 i minnet och startas i real-mode.

GNU/Linux, nämligen LILO, vilket betyder *LInux LOader*, och är skrivet av Werner Almesberger respektive *GNU GRUB* vilket utläses *GNU GRand Unified Bootloader* och är ett universellt bootprogram, ursprungligen skrivet av Erich Boleyn. Dessa använder visserligen huvudbootblocket på samma vis som DOS och Windows, men startandet av GNU/ Linux-partitionerna sköter de efter eget gottfinnande, t.ex. struntar de i vilken partition som markerats som aktiv och startar direkt ett sekundärt program från någon egendefinierad partition, som i sin tur startar ett operativsystem från någon annan partition. De lagrar dock partitionstabellen (se nästa avsnitt) på ett vis som är kompatibelt med andra, inte fullt lika förslagna operativsystem.

Det är möjligt att med både LILO och GNU GRUB åstadkomma konfigurationer så att mer än ett operativsystem kan startas från din dators hårddiskar. Om du kan starta *två* olika operativsystem kallas detta *dual boot*, men alla konfigurationer som kan starta mer än ett operativsystem kan lika gärna kallas *multiboot*, i synnerhet bör du använda det begreppet om fler än två operativsystem kan startas. Det existerar ingen egentlig standard för hur multiboot skall gå till, och utvecklarna av GRUB har därför skrivit en: *the multiboot specification*. (Se [22].) Frågan är om andra operativsystemtillverkare kommer att följa denna specifikation, eller ens ägnar frågan särskilt stort intresse.

Om du vid installationen av din distribution behöver välja mellan att använda LILO eller GNU GRUB är valet enkelt: GRUB. Detta program är betydligt mer sofistikerat än LILO, som i princip bara föredras av de som använder det av gammal vana. GRUB är dessutom betydligt enklare att konfigurera. Om du inte kan initiera GRUB via ett installationsprogram så är principen denna:

```
# grub
grub> root (hd0,0)
grub> setup (hd0)
```

Första raden startar GRUB, andra raden talar om för GRUB att du vill boota från primära diskens (hd0) första partition. (Listigt nog numrerad så att första partitionen får nummer noll.) Den tredje raden ber GRUB att installera sig självt i huvudbootblocket på den primära disken. Du måste därefter även skapa en konfigurationsfil till GRUB på den plats där du valt att installera det: de flesta väljer att installera GRUB i **/boot** (många lägger också /boot på en egen partition, se nedan) och då skall konfigurationsfilen heta **/boot/grub/grub.conf**. Om GRUB är ordentligt installerat skall det finnas en exempelfil som heter **/boot/grub/grub.conf.sample** att utgå ifrån. Kopiera och använd denna som förebild.

Om du tänker installera ett annat operativsystem förutom GNU/Linux — typiskt Microsoft Windows — bör du installera detta *först* (och lämna då plats åt GNU/Linux, se nedan) och GNU/Linux *efteråt*. Anledningen till att det skall göras i denna ordning är att LILO/GRUB är mer flexibla, och kräver sitt eget huvudbootblock, medan andra operativsystem, typiskt Windows, gärna får för sig att skriva över huvudbootblocket med sin egen favoritkod utan att fråga.

När ditt installationsprogram för en viss GNU/Linux-distribution frågar om det skall skriva över huvudbootblocket, skall du normalt svara *ja*, och sedan låta LILO eller GRUB starta andra eventuella operativsystem på din dators hårddisk. Du kan eventuellt nöja dig med att skapa den startdiskett som de flesta installationsprogram kommer att propsa på att du behöver, och därefter starta GNU/Linux från den disketten. Detta är inte särskilt slugt, låt bli om du inte måste. Startdisketten skall dock skapas och användas i prekära situationer (även om en bootbar installationsskiva för det mesta duger lika bra i dessa trångomål).

Kanske vill du göra en kopia av ditt huvudbootblock, det går i så fall bra att skapa detta med **dd if=/dev/hda of=MBR bs=512 count=1**, vilket kopierar huvudbootblocket från **/dev/hda** till en fil med namnet **MBR**. Om programmet **hexdump** finns installerat i ditt operativsystem kan du titta närmare på innehållet med **hexdump MBR**. Om du är nyfiken på hur Linuxkärnan sedan startas upp av LILO eller GRUB så kan du bläddra fram till avsnitt 5.4.2 på sidan 184.

Utöver LILO och GRUB finns för mycket modern hårdvara även **ELILO**. Detta är en version av LILO som skrivits om för att stödja den nyare varianten av BIOS, *Extensible Firmware Interface* (EFI). Om du har en hårdvara som använder detta[17] bör du även se till att hela din distribution är anpassad för det. GRUB är i dagsläget ännu inte anpassad för EFI.

4.1.4 Partitionering

En hårddisk kan *partitioneras* i olika logiska enheter. Detta innebär att delar av en och samma hårddisk ur operativsystemens synvinkel ser ut som separata enheter. Varje sådan separat enhet kallas för en *partition*. En vanligt förekommande missuppfattning är att partitionering har något med formatering att göra, detta är felaktigt, varje partition får sedan formateras om den skall innehålla ett visst filsystem.[18]

[17]T.ex. använder de nyaste Intel x86-baserade MacIntosh-datorerna uteslutande EFI.

[18]Under 1980-talet och tidigt 1990-tal fanns hårddiskar som ibland krävde en s.k. ”lågnivåformatering”, vilket innebar att MFM-mönstret (kodningen av ettor och nollor på

Detta sköts på olika vis i olika datorarkitekturer, men på IBM PC finns idag möjlighet att använda maximalt fyra olika grundpartitioner på en viss hårddisk eller hårddisk-liknande enhet.[19] Informationen om varje partition lagras i en partitionspost. Dessa sammanlagt fyra poster utgör den information som kallas *partitionstabell* och lagras som nämnts i den första hårddiskens huvudbootblock. (Se figur 4.2 för att se hur partitionsinformationen lagras i huvudbootblocket.)

Varje partition har en *typ*, vilket i praktiken bara är en siffra. *En* av partitionerna skall markeras som *aktiv* eller *primär*. (De övriga kallas vanligen för *logiska partitioner*.)

Eftersom moderna operativsystem ofta kräver betydligt fler än fyra partitioner uppfanns tidigt en ”magisk” partition med typen 5, vilken kallades *utökad partition*. Denna pekar i förekommande fall ut en plats där ytterligare fyra partitioner kan lagras (utanför huvudbootblocket) i form av en egen partition, d.v.s. det är rent fysiskt bara en väldigt liten partition, som innehåller en partitionstabell för maximalt fyra partitioner. Denna placeras efter de tre första ”normala” partitionerna. Den sista av dessa fyra nya partitioner kan också vara av typ 5, o.s.v. så att ett obegränsat antal partitioner kan skapas i en form av länkad lista.[20]

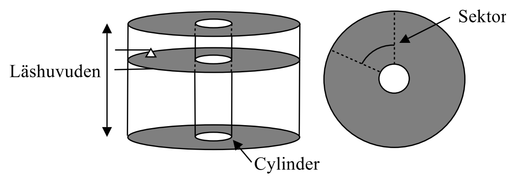
Här skall jag som hastigast nämna något om hur hårddiskar fungerar. Dessa anger en position på hårddisken med tre siffror: *cylinder*, *sektor* och *läshuvud*. Inuti en hårddisk sitter ett antal skivor. Varje skiva läses med två läshuvuden: ett på ovansidan och ett på undersidan. Varje skiva har två egna huvuden. Alla huvuden rörs simultant över skivornas yta. Positionen längs skivornas radie kallas för en *cylinder*, eftersom skivorna då de roterar med hög hastighet och läses av på alla huvuden samtidigt får formen av en ”informationscylinder” med magnetiskt lagrad data. Om vi betraktar ena ytan av en enda skiva kallar vi detta för ett *spår* (engelska: *track*) där läshuvudet kan gå fram. På varje spår finns ett antal *sektorer* (vilket har exakt samma betydelse som i geometrin), och varje sektor är 512 byte stor. Som du säkert inser kan alltså varje position på hårddisken i tre dimensioner anges med dessa

---

själva skivan) preparerades för alla spår och sektorer. Sådant sker numera enbart i fabrik och berör inga slutanvändare. På den tiden lågnivåformateringar gjordes måste detta göras innan själva partitioneringen, vilket kanske har bidragit till sammanblandningen av partitionering och formatering.

[19] Även s.k. USB-drivar och helt vanliga disketter brukar hanteras som om de vore ”vanliga” hårddiskar och har därför vanligen ett huvudbootblock och en (ensam) partition. Huvudbootblocket på disketterna var för övrigt under 1990-talet en vanlig infektionsväg för diverse datorvirus.

[20] På grund av denna klumpighet har den nya IA64-arkitekturen ett annorlunda partitioneringssystem som heter *Extensible Firmware Interface* (EFI) varom litet eller inget är känt av författaren, men som säkerligen kommer att bli vikigt att kunna hantera i framtiden.



Figur 4.3: Här ser vi till vänster läshuvudenas läge i y-led. Cylindrarna som tänkta snitt genom alla skivor, och till höger en skiva med en enstaka sektor av ett spår markerad.

tre värden: läshuvud, cylinder och sektor.[21] Kanske kan figur 4.3 öka den intuitiva känslan för hur detta fungerar en aning.

Du behöver normalt inte bekymra dig om huvuden, cylindrar och sektorer när du installerar ditt system, men i händelse att du vill åstadkomma något utöver det vanliga är det viktigt att känna till ungefär hur hårddisken och bootsektorerna hanteras. Visssa versioner av BSD kräver exempelvis att du anger installationspartitionens plats i form av läshuvud, cylinder och sektor.

Detta något tillyxade system används av de allra flesta operativsystem, inklusive GNU/Linux bootstrap loaders såsom LILO och GNU GRUB. Dock kommer de inte att respektera uppgiften om vilken partition som är primär, eller bry sig om att Microsoft DOS och Microsoft Windows bara kan hantera boot från grundpartitionerna och inte de utökade. Istället presenterar de oftast en lista över tillgängliga operativsystem och erbjuder användaren att välja vilket som skall startas, så som beskrevs i föregående avsnitt.

En distribution behöver vanligen ett flertal hårddiskpartitioner att installera sig på. Om du har ett opartitionerat område på din hårddisk brukar det inte vara så svårt att installera distributionen där: de flesta moderna distributioner kan automatiskt partitionera och formattera detta utrymme på ett sätt som passar distributionen. Det utrymme som krävs brukar variera beroende på hur mycket av din valda distribution du vill lägga in, men minimistorleken brukar i dagsläget ligga runt en gigabyte.

Om du redan har ett annat operativsystem installerat på din dators hårddisk står du inför ett problem. Givetvis kan du ta bort det gamla operativsystemet helt, och detta är också det enklaste alternativet. Om det andra operativsystemet har lämnat tomt, opartitionerat utrymme

[21] Verkliga hårddiskar är numera ganska sofistikerade och följer kanske inte dessa angivelser alls, men den bärande tanken är densamma, så även om hårddisken internt översätter dessa angivelser till något helt annat så används ändå denna adressering.

på hårddisken, kan du såklart installera GNU/Linux där.

Ett alternativ som inte alla tänker på är att köpa en extra hårddisk — den behöver inte vara speciellt stor, c:a sex gigabyte räcker ju gott och väl — för att starta GNU/Linux på, och installera denna i din dator. Dock kan det bli problem om du vill kunna starta operativsystem från flera olika hårddiskar, till skillnad från flera olika partitioner: inte alla datorers BIOS är gjorda för detta. GRUB eller LILO brukar dock kunna reda ut detta problem under förutsättning att de installeras i den primära hårddiskens huvudbootblock.

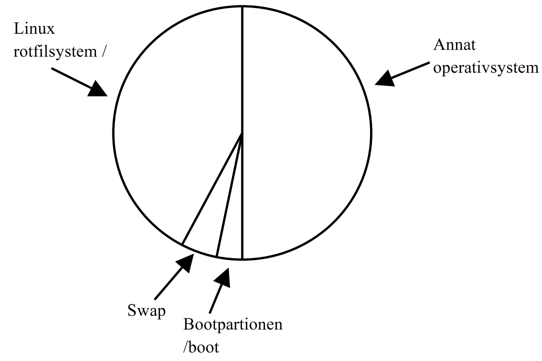
Om du har ett annat operativsystem som i och för sig har lagt beslag på hela hårddisken, men som inte använder allt utrymme, brukar du normalt önska att du kunde göra en ickedestruktiv ompartitionering, så att innehållet i de partitioner som redan finns på hårddisken inte skadas. Detta är också möjligt: programmet *GNU Parted* klarar detta. Eftersom Parted är gjort för att köras under GNU/Linux krävs dock Linux. Detta moment 22 löser de flesta genom att starta GNU Parted från en startdiskett eller CD-ROM som innehåller en minimal Linuxkärna. En vanlig lösning är att använda *GParted LiveCD*[22] som innehåller det helt grafiska *GParted* eller *System Rescue Cd*[23] som innehåller *QtParted*. Båda dessa program använder i sin tur GNU Parted och **ntfsresize** (se vidare nedan). GParted och QtParted är skrivna för att efterlikna Windows-programmet Partition Magic.

GNU Parted kan i dagsläget inte hantera NTFS vilket är det filsystem som används av Microsoft Windows 2000, Windows XP och senare varianter. Istället används programmet **ntfsresize** för detta. Det programmet är i sin tur helt fristående, men baserat på samma källkod som NTFS-drivrutinerna för Linux-kärnan och utvecklat av samma personer.[24]

För att utföra ompartitioneringsmanövrar fordras naturligtvis att du vet vad du håller på med. Gör inte detta om du är osäker på slutresultatet! En annan sak som många glömmer är att göra backup på hårddisken *innan* de ompartitionerar. Gör det! Om du vill göra en avbild av hela hårddisken som du sedan kan återställa kan du använda programmet *Partimage*,[25] ett fritt program som fungerar som det populära programmet *GHOST*. Även detta program finns på System Rescue CD.

När du sedan installerar din valda GNU/Linux-distribution kommer du att få frågor om partitionering. Ofta kan du välja att låta installationsprogrammet partitionera det lediga utrymmet åt dig. Om detta

[22] Se: `http://gparted.sourceforge.net/livecd.php`
[23] Se: `http://www.sysresccd.org/`
[24] Se: `http://linux-ntfs.sourceforge.net/`
[25] Se: `http://www.partimage.org/`



Figur 4.4: En illustration av hur hårddisken kan partitioneras upp i en dator. En hårddisk partitioneras dock inte i sådana här cirkelsektorer, utan i "cylindrar", vilket är koncentriska cirklar med ökande radie.

inte är möjligt kan du vara tvungen att bestämma saker som storlekar på rotpartition, och hur stor *swappartition* du vill skapa. Den senare används av Linuxkärnan för att växla ut minnessidor (engelska: paging) när/om det fysiska minnet tar slut och bör enligt en tumregel väljas till den dubbla storleken av datorns internminne, d.v.s. har du 256 megabyte internminne i din dator skall den göras 512 megabyte stor. De partitioneringsprogram som används vid installationen brukar vara **fdisk** (vilket inte på något vis är DOS-programmet med samma namn, men snarlikt i funktionalitet), eller Red Hats **disk druid** vilket är ett lite hjälpsammare program. Modernare distributioner har partitioneringsverktyget inbyggt i installationsprogrammet.

Det finns tre andra partitionsprogram utöver dessa: **sfdisk** är ett program som liknar **fdisk** i det mesta och **cfdisk** är ett menystyrt, textbaserat och ganska lättanvänt program som bör föredras framför **fdisk** om det finns tillgängligt. (Det föredras bl.a. av Debian.) GNU Parted (körs med kommandot **parted** kan också användas för all form av partitionering, så summa summarum finns det inte mindre än fem olika partitioneringsprogram för att göra i stort sett samma sak. **cfdisk** och **disk druid** är lite mer grafiska, medan GNU Parted är lite mer kraftfullt.

4.2 Debian GNU/Linux

Officiell hemsida: `http://www.debian.org/`

Debian GNU/Linux är en av de få distributioner som verkligen kallar

distributionen för *GNU/Linux* snarare än ”bara Linux”. Efter att projektet startades 1993 har en hel del hänt med Debian. Den huvudsakliga filosofin bakom projektet presenteras i *Debian Social Contract*[8], och innebär i korthet:

- Att Debian *använder fri mjukvara* enligt Debian Free Software Guidelines, vilka i sin tur är i princip samma definition som används av Free Software Foundation och godkänner såväl BSD-licensen som GNU GPL och Perl och Apache:s artistiska licenser.

- Att Debian *stödjer fri mjukvara*, genom att återföra information till de ingående projekten, hjälper dessa att hitta och rätta till buggar (vilket av deltagarna kallas *”upstream”*), och till sist:

- Att projektet är *helt öppet*, även rörande säkerhetsbrister och andra fel. All information om Debians interna organisation och göranden är publik, inga beslut fattas i det fördolda.

Alla användare kan om de vill delta i Debianprojektet, d.v.s. utvecklingen av Debian drivs inte slutet, så som ofta är fallet i distributionsföretagen, utan är lika öppet för nya deltagare som GNU och Linux är.

Arbetet med Debian delas rent konkret mellan utvecklarna på så vis att olika personer är huvudansvariga (engelska: maintainer) för olika *paket* vilka är av varierande komplexitet och viktighet. Exempelvis är Linuxkärnan och fönstersystemet *X* exempel på väldigt stora, komplexa och viktiga paket, medan däremot paketet med skämtsamma manualsidor[26] inte är särskilt viktigt. Utöver detta har Debian en omfattande

[26] Detta paket heter **funny-manpages** och innehåller uppslagsord som *rtfm*, *sex* o.s.v.

beslutande organisation för policyfrågor och liknande, dit olika personer väljs med utgångspunkt från meriter. Ytterst ansvarig är projektledaren, vilket ursprungligen givetvis var Ian Murdock, men som när detta skrivs för tillfället är Branden Robinson. Det finns en fullständig konstitution utarbetad för hur alla frivilliga deltagare i Debianprojektet skall samarbeta, och ett omfattande policydokument som beskriver hur utvecklarna skall sköta paketen. Bland utvecklarna finns flera svenskar.

För att säkerställa autenticiteten i paket som laddas ned från projektet signerar utvecklarna sina paket med GPG-kryptonycklar, och signaturerna verifieras sedan vid installationen. På så vis kan du lita på att Debianpaketen verkligen är en del av projektet. Utvecklarna har signerat varandras nycklar korsvis för att bygga upp den ömsesidiga tillit som krävs.[27]

Distributionen består av tre olika deldistributioner: *unstable*, *testing* och *stable*. När en utvecklare just har gjort ett paket färdigt placeras det i *unstable*-distributionen. När det testats där i 10 dagar och portats till alla de datorarkitekturer som stödjs av Debian flyttas det till *testing*-distributionen där det testas ytterligare. När en *testing*-distribution anses tillräckligt stabil byter den namn till *stable*.[28] Varje *testing*-distribution har ett kodnamn från filmen Toy Story (t.ex. *Potato*, *Woody*, *Sarge* etc.) men när den slutligen släpps får den istället ett versionsnummer, t.ex. Debian GNU/Linux 3.1 (som tidigare hette *Sarge*).

Det är här du kommer att upptäcka att Debian är en distribution med fokus på stabilitet: det tar evigheter för *testing* att förvandlas till *stable*. Debians operativsystemrevisioner är ökänt segdragna och kan ta bortåt två–tre år. Givetvis är det då möjligt att trots allt köra *testing*-versionen: det ger ingen större skillnad för en normal användare, och är mer à jour med utvecklingen.

Var och en av deldistributionerna *unstable*, *testing* och *stable* innehåller sedan tre underkataloger för paketkategorierna *main*, *non-free* och *contrib*: den fria mjukvaran finns i *main* men därutöver tillhandahåller Debian två kataloger med presumptivt icke-fri mjukvara, kallade *contrib* och *non-free*. Skillnaden mellan *contrib* och *non-free* är att contrib är fri programvara som *i sin tur* är beroende av ofri programvara. De användare som vill kan helt välja bort denna typ av mjukvara. Existensen av dessa kataloger är inte särskilt omtyckt av Free Software Foundation, men de som arbetar i projektet tycker själva i allmänhet att detta är en bra kompromiss. *main* är den katalog som innehåller det egentliga Debian och består enbart av fri programvara, medan de andra är att be-

[27] I dagsläget fungerar detta kanske inte riktigt som det ska. Paketen signeras innan de skickas till byggservrar, där paketen för de olika arkitekturerna byggs. Resultatet från byggena är i dagsläget helt osignerat.

[28] Under en kortare period har denna namnet *frozen*, men blir sedan slutligen *stable*.

trakta som ”extra godis” (eller som Richard Stallman säkert skulle säga: förgiftat godis).

Som nämndes i förbigående stödjer Debian en uppsjö datorarkitekturer: i dagsläget förutom de vanliga Intel och AMD-baserade PC-maskinerna[29] även de flesta gamla Macintosh-datorer, Digital Alpha, Hewlett-Packards arbetsstationer, Amiga, olika MIPS och PowerPC-baserade system, Sun SPARC och stordatorn IBM S/360. Flera av dessa är Debian ensamt om att stödja.

När det gäller hjälp och support är tanken med Debian framför allt att användarna skall hjälpa varandra. På webbsidan `http://lists.debian.org/users.html` finns en rad epostlistor för olika kategorier av slutanvändare, och för svenskar passar listan *debian-user-swedish* som hand i handske. Om du tänker använda Debian mycket bör du gå med i denna epostlista. Det finns även företag som säljer support för Debian, men kanske inte till enstaka slutanvändare, utan till andra företag och organisationer. På sidan `http://www.debian.org/consultants/` finns en lista över sådana konsultföretag över hela världen, varav i dagsläget 14 är svenska. Konsulttjänster av det här slaget är ganska dyrt och inget en hemanvändare har råd med, men för ett företag kan sådan hjälp naturligtvis vara väl värt pengarna.

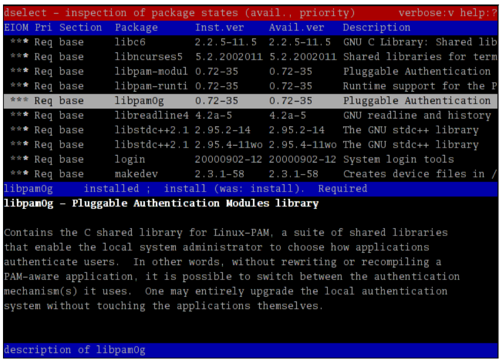
Det skall också nämnas att Debianprojektet är progressivt på många vis, till exempel är det teoretiskt sett inte ens begränsat till Linuxkärnan: i dagsläget pågår tester med såväl FreeBSD-kärnan som GNU Hurd-kärnan.

4.2.1 Installation av Debian

Installationsprogrammet i Debian kan upplevas som något gammaldags både för nybörjare och för den som tidigare använt exempelvis Red Hat Linux. Det är inte grafiskt och innehåller idiomatiska inslag, även om det radikat förbättrades i och med version 3.1 av Debian. Däremot har det goda konfigurerbarhetsegenskaper, eftersom bara en absolut minimal ”bottenplatta” av GNU/Linux-komponenter installeras till att börja med.

Efter detta startas *Debian Task Installer* som gör det möjligt att välja ut hela grupper av paket som är gjorda för att fungera tillsammans. Du väljer här exempelvis ”skrivbordsmiljö” om du vill ha en fungerande skrivbordsmiljö med fönstersystemet *X*, GNOME och/eller KDE. Detta ger en ganska grovkornig kontroll över systemet och om du vill styra det hela mer i detalj mer möts du av ett program som heter **dselect** (Debian package selection tool) och där du förväntas välja vilka paket

[29]Hit hör naturligtvis även 64-bitaras IA64 och Opteron-baserade PC-datorer.



Figur 4.5: Här syns det något skräckinjagande programmet dselect som används för att välja paket under Debian-installation.

du vill ha installerat på din dator, genom att vandra runt i menyer med piltangenterna.

Det är inte enkelt att konfigurera brandväggsinställningar och annat som egentligen borde göras under installationen. Däremot är säkerhetstänkandet högt: om du har en Internetförbindelse kommer Debian automatiskt att kräva att få ladda ner och installera mängder av säkerhetsuppdateringar för att se till att ditt system är ordentligt säkert.

Under hela installationsprocessen fullkomligt *kräks* programmen ur sig komplicerade tekniska frågor om både det ena och det andra, som till exempel om fontrenderaren FreeType2 skall användas i Mozilla, vilket en novis kanske blir ganska rejält förvirrad av. Den avancerade användaren tycker däremot säkert att det är trevligt att få skruva på alla dessa små inställningar.

Efter att all installation är avslutad kommer inte grafiken att fungera över huvud taget, även om du valt en skrivbordsinstallation. Det krävs först att du manuellt installerar och konfigurerar delar av fönstersystemet *X*, vilket i sin tur kräver ytterligare specialkunskaper.[30] Att få *X* att fungera under Debian kan vara ganska jobbigt och det är ibland enklare att lita till egna kunskaper om *X*-servern än att försöka använda Debians konfigurationsverktyg.

En viss bot på Debians svårinstallerbarhet kommer förmodligen i och med att företaget Progeny har tagit sig för att konvertera Red Hats installationsprogram *Anaconda* så att det kan användas även för Debian. I dagsläget är detta dock mest ett experiment.

[30] **dpkg-reconfigure xserver-xfree86** *kan* ge en *viss* hjälp.

4.2.2 DEB, dpkg och APT-systemet

Debian har ett eget och ganska unikt paketeringssystem som också är det äldsta. Detta system bygger på programpaket som kallas DEB-paket, och vilka normalt har filändelsen *.deb*. Paketen ligger installerade på flera platser i filsystemet, mestadels under */usr* men också direkt i roten, t.ex. i */bin*. I princip tillhör de flesta filer i systemet ett paket, men innehållet i */home*, */usr/local*, */proc*, */tmp*, och ändrade konfigurationsfiler gör det exempelvis *inte*. En central databas håller reda på vilka paket som är installerade samt vilka filer som hör ihop med vilka paket.[31] På så vis är det enkelt att lägga till och ta bort paket utan att skada andra installerade program.

Paketfilerna har följande egenskaper:

- De är kompilerade för en och endast en datorarkitektur. Debian stödjer som sagt flera olika arkitekturer. Eventuellt har Debian-utvecklarna också ändrat lite i programmen, så det är inte helt säkert att resultatet blir exakt detsamma som om du kompilerat och installerat programmet helt på egen hand.
- De innehåller ofta en del specialskript som körs före och efter det att själva filerna som ingår i paketet har installeras. Dessa skript kan få för sig att ställa frågor under såväl installation som borttagning, och då får du lov att svara på dem. Frågorna kan vara kryptiska och svårbegripliga, men då presenteras ofta något standardiserat svarsalternativ som du kan välja om du skulle vara osäker.

Paketfilerna har namn i stil med *foo_2.0-12.deb* vilket skall utläsas som version 2.0 av paketet *foo*, och 12 är revisionen av själva paketet.[32]

För manipulation, tillägg och borttagning av DEB-paket används följande kommandon:

dpkg är den universella schweiziska armékniven för DEB-paket.

dpkg --list plockar fram en lista över alla paket som är installerade på Debiansystemet. Vill du bara se vissa kan du ange det med ett reguljärt uttryck, exempelvis **dpkg --list 'foo*'**, så syns bara paket vars namn börjar med *foo*.

dpkg --install foo_2.0-12.deb installerar det angivna paketet. Innan paketet installeras kontrollerar dpkg ifall något annat paket måste installeras först, eller om paketet är beroende av att en annan version av något paket skall vara installerat. (Då kan det bli problem.)

[31] Denna databas ligger i */var/lib/dpkg*.

[32] Se vidare avsnitt 3.5, sidan 126 om versionsnummer.

dpkg --remove foo tar bort det angivna paketet. Nu behövs inga filnamn, eftersom det abstrakta paketet inte har något att göra med vad filen det installerades ifrån hette.

När du tar bort ett paket raderas inte inställningar för paketet; det beror på att du kanske bara tänkt dig att ta bort paketet för att installera ett nyare, och då vill du ju typiskt behålla konfigurationen du hade innan. För att radera även konfigurationsfilerna (vilka då normalt finns i */etc*) skriver du **dpkg --purge foo**.

dpkg -L foo ger en lista över alla filer som paketet *foo* har installerat i ditt operativsystem.

dpkg -p foo ger detaljerad information om paketet *foo*. Exempelvis vem som gjort paketet och vilken arkitektur det är avsett för, samt vilka beroenden (krav på andra paket) det har.

dpkg -S /usr/bin/foo talar om vilket paket den där filen */usr/bin/foo* kom ifrån. Fungerar även med globbning, **dpkg -S /usr/bin/fnord*** t.ex.

dpkg-reconfigure foo kör konfigurationsskriptet från paketet *foo* en gång till. Om du skrev fel eller av annan anledning vill göra om paketkonfigurationen (om det finns någon sådan) är detta din hjälp i nöden. Typexempel: **dpkg-reconfigure xserver-xfree86**.

Normalt vill du inte använda dpkg direkt. Använd apt-get (se nedan) istället.

dselect nämndes redan under installationsbeskrivningen, detta program vars utseende återfinns i figur 4.5 kan inte anses vara intuitivt av någon människa annat än Debianextremister. Det enda dselect egentligen gör är att göra en lista över operationer som skall utföras av dpkg, som sedan gör det faktiska jobbet. De kryptiska tecknen " ***" längst ut till vänster under rubriken **EIOM** skall utläsas

E rror — en stjärna (*) i denna kolumn anger att paketet är trasigt (vilken tur att inget paket är trasigt i figuren). Ett **R** istället för en stjärna anger att det är *väldigt trasigt* och måste installeras om.

I nstalled — en stjärna (*) anger att paketet är installerat. Andra tecken: **-** (bindestreck): programmet har varit installerat, det är borttaget men konfigurationsfilerna som hör till programmet är däremot inte borttagna. Bokstäverna **U**, **C** och **I** bety-

der att paketet är trasigt på olika vis och antagligen behöver installeras om.

O ld — vad som stod angivet för detta paket i kolumnen **I** innan vi startade dselect och började pilla runt.

M ark — vad du tänker dig att göra med paketet härnäst: *****: installera eller uppgradera paketet, **-**: ta bort paketet, **=**: gör ingenting, **_**: rena (purge), d.v.s. ta bort både paket och konfiguration, **n** detta paket är nytt och du får lov att bestämma dig för vad du vill göra med det.

Personligen anser jag att dselect går att leva utan, undantaget under själva installationen (tyvärr) då en mängd paket måste processas på en gång. Det kanske mest störande med dselect är dock hjälpsystemet som kräver saker som att du skall välja hjälpmenyer genom att trycka på bokstavskombinationer och avsluta program genom att trycka på punkt (.).

apt-get är ett program inom konceptet *Advanced Package Tool*, APT[33], och är ett centralt, bra, och mycket viktigt program: det laddar ned och installerar den senaste versionen av ett program som finns i en självunderhållande lista över tillgängliga program. Det har också förmågan att lösa upp konflikter i de fall att olika paket är beroende av varandra eller inte kan samsas av andra skäl. Ibland kallas detta för att APT-sviten är ett *beroendehanteringsverktyg*. Förutom de paket som tillhandahålls av Debian själva finns det andra listor som du kan välja att plocka hem paket ifrån.

Den enda nackdelen är att en hyfsat snabb nätverksförbindelse krävs för att detta skall fungera någotsånär kvickt. Naturligtvis använder även apt-kommandona dpkg i slutänden, men apt tillför möjligheten att hålla sig automatiskt uppdaterad. Alla Debiananvändare lär sig snart följande kommandon:

apt-get update uppdaterar listorna över tillgängliga program från de källor som angivits i filen */etc/apt/sources.list*. Detta skall du normalt skriva först.

apt-get install foo laddar ned och installerar senaste versionen av paketet *foo*. Kommandot kommer även att kontrollera ifall ett annat paket måste installeras först för att det hela ska fungera, och kommer i så fall att ladda ner och installera dem också, eventuellt efter att först ha frågat om lov. Enklare blir det inte.

[33]Strikt taget är APT i sig ett programbibliotek skrivet i C++, som andra program, däribland **apt-get** och Synaptic använder sig av.

apt-get upgrade laddar ner och installerar senaste versionen av alla paket som har en tillgänglig uppgradering. Utomordentligt för att hålla programmen fräscha och säkerhetspatchar installerade.

apt-get remove foo tar bort paketet *foo* från systemet. Samma funktion som **dpkg --remove foo**. Inställningsfiler som tillhör paketet kommer inte att tas bort.

apt-get dist-upgrade är som upgrade men avsedd för mer prekära situationer då inte alla program kan uppdateras. APT kommer då att själv välja mellan olika paket för att genomföra de viktigaste uppgraderingarna på bekostnad av mindre viktiga. Typiskt användningsområde: att byta från *stable* till *testing*: byt ut källorna som pekas ut i */etc/apt/sources.list* (med **apt-setup** om du vill) och skriv **apt-get update** för att läsa om paketkatalogerna, skriv sedan **apt-get dist-upgrade** för att byta till *testing*. Namnet på funktionen kommer antagligen från detta sätt att byta distribution.

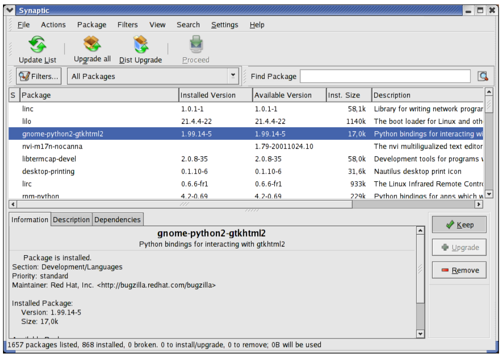
Notera att det räcker med att byta pekare i */etc/apt/sources.list* för att en helt annan version av Debian skall börja användas. På ett liknande vis innebär en pekare till *stable* eller *testing* att du förr eller senare kommer att tvångsuppgraderas till en ny version av Debian en vacker dag då du kör **apt-get upgrade**. (Du märker detta på att APT kommer att fråga om den får lov att byta ut några hundra paket.) För att undvika att detta händer kan du peka ut en plats med din distributions namn istället för *stable* eller *testing*: använd då namnen *potato*, *woody*, *sarge* o.s.v.

apt-get source foo laddar ned källkoden för paketet *foo* om nedladdningsplatser för källkod finns angivna i *sources.list*.

apt-setup hjälper dig att ställa in vilka källor du skall plocka hem alla paket ifrån. Välj en server som ligger i Sverige. Körs normalt bara enstaka gånger.

aptitude är ett annat textbaserat installationsprogram. Det liknar dselect i sin obegriplighet och kryptiska utformning, men är inte fullt lika illa, och använder dessutom **apt-get** för att plocka hem program från listor av tillgänglig programvara.

synaptic är en användarvänlig och grafisk motsvarighet till aptitude, byggd på GTK+. Använd detta program istället för dselect om du har fönstersystemet *X* startat. Är det inte redan installerat så installera det med **apt-get install synaptic**. (Se figur 4.6.)



Figur 4.6: Synaptic är det förmodligen mest lätthanterliga pakethanteringsprogrammet för APT-systemet.

Det finns ännu fler funktioner i APT, men de angivna är de klart viktigaste. APT har blivit så populärt att det t.o.m. finns portat till Red Hat Linux, där det används med RPM-systemet.

För att manipulera paketdatabasen och filsystemet krävs att du är inloggad som root-användaren i ditt Debiansystem. Du kan dock som vanlig användare åtminstone undersöka vilka paket som är installerade.

Tack vare dels den höga stabiliteten och säkerheten i Debian, samt det lättskötta APT-systemet för systemunderhåll och säkerhetsuppdateringar, anses det vara ett mycket bra system för servrar som skall installeras och sedan fjärradministreras. Du vill förmodligen inte gärna råka ut för att en säkerhetsuppdatering som du installerar kraschar en dator som befinner sig på andra sidan jordklotet, och med Debian är den risken minimal. Även som vanlig skrivbordsdator är Debian-distributionen populär, men då använder många *testing*-versionen för att hålla mjukvaran någotsånär aktuell. Du byter med **apt-get dist-upgrade** som beskrevs ovan.

4.2.3 Felrapportering

Om du hittar något fel i Debian, även om det kanske inte beror på Debian i sig utan på underliggande mjukvaror, så skall du som god projektdeltagare rapportera felet. Fel i Debian rapporteras för närvarande via epost, i ett egensnickrat felhanteringssystem som finns beskrivet på `http://www.debian.org/Bugs/`. Ofta får du feedback snabbt på rapporterade fel, och ombeds då att verifiera att felet är löst i *testing*-



distributionen, om du har den.

Tänk på att felrapporteringen är avsedd för verkliga *fel*. Behöver du bara hjälp med att begripa hur något fungerar ska du fråga på epostlistan istället.

4.3 Red Hat Linux / Fedora Core

Officiella hemsidor:
`http://www.redhat.com/`
`http://www.fedoraproject.org/`

Den första rent kommersiella GNU/Linux-distributionen som någonsin tillverkades var Yggdrasil Linux, framställt av Yggdrasil Computing med Adam Richter som huvudman. När denna distribution började säljas på en CD-ROM-skiva för $99 under 1993 ändrades läget radikalt för fri mjukvara. Detta visade för första gången på allvar att det faktiskt gick att tjäna pengar på fria mjukvaror.

Samma år grundade Marc Ewing firman Red Hat, namngiven efter hans fars skärmmössa (något i stil med en baseballkeps) som var röd och vit, som ränderna i den amerikanska flaggan. Mössan hade han använt på college och sedan slarvat bort, så företaget döptes till Red Hat för att hedra den borttappade mössan. Den andra huvudsakliga initiativtagaren till Red Hat var kanadensaren Robert ”Bob” Young, finansiären bakom bl.a. tidningen *Linux Journal*. Youngs företag ACC Corporation slogs år 1995 samman med Red Hat till ett gemensamt företag, varvid Young tog rollen som VD. I november 1995 utökades företaget ännu en gång genom att gå samman med Cygnus Solutions, ett företag som specialiserat sig på GNU-projektets kompilatorer och utvecklingsverktyg för bland annat inbyggda system. I dagsläget har företaget runt 700 anställda på 22 olika orter runt om i världen, bland annat USA, Australien, England och Japan.

Huvuddelen av finansieringen till detta mellanstora företag kommer från konsultverksamhet. Red Hat tjänar pengar på att hjälpa andra företag att byta till Linux från andra operativsystem, reguljär IT-drift, utbildning av Linuxtekniker samt avancerat användarstöd och konsultation för utveckling av inbyggda system.

Distributionen *Red Hat Linux*, som en gång var anledningen till att företaget startades är numera nedlagd till förmån för projektet *Fedora Core* (se nedan). Gamla Red Hat Linux var avsedd för vanliga IBM PC-slutanvändare, men företaget tillverkar nu endast olika former av *Red Hat Enterprise Linux*, som är avsedd för såväl klienter som serverdrift av stora system (i så fall typiskt Oracle och SAP R/3), och som säljs dyrt i form av en halvårsprenumeration som inkluderar installationsmedia[34] och med support (f.n. finns denna support i England för svenska kunder). Det "vanliga" Red Hat Linux vars sista version var Red Hat Linux 9 finns dock fortfarande tillgängligt gratis.

Bland användare upplevs Red Hat Linux och Enterprise Linux väldigt ofta som en *produkt*, och användaren känner sig som en *kund*, till skillnad från Debian GNU/Linux som upplevs som ett *projekt* där användaren är *deltagare*. Red Hat profilerar sig mot storanvändare: typiskt företag med tusentals datorer. Andra användningsområden är stora serverparker och kluster för storskalig beräkningskapacitet. Detta till trots upplever många användare Red Hat Linux som en utomordentlig "hemmaprodukt".

Den första versionen av Red Hat Linux kom ut den tredje november 1994 och kostade då $49.95 på CD-ROM. Sedan dess har företaget introducerat nya versioner c:a 1 gång per år, räknat efter huvudversionsnumret. I och med version 9 slutade Red Hat helt att tillverka Red Hat Linux för hemmabruk, och fokuserade all verksamhet på Red Hat Enterprise Linux, som för närvarande är uppe i version 3.

Under en period (runt huvudversion 6 och 7) ansågs Red Hat Linux vara ganska instabil och program som ingick i distributionen kraschade tyvärr ofta. Sedan dess har stabilitetsproblemen ordnat till sig. Red Hat Enterprise Linux har sedan utvecklingen på denna variant startades varit inriktad mot hög stabilitet.

Utvecklingen av Red Hat Linux sköttes fram till sensommaren 2003 på ett traditionellt vis: företaget tillverkade allt innanför sina egna väggar, och släppte inte in utomstående utvecklare annat än som felrapportörer. Vid denna punkt slogs projektet samman med ett fristående initiativ kallat *Fedora Core*, och blev ett öppet projekt, dit alla användare

[34]Du kan säkert få tag på Red Hat Enterprise Linux i form av såväl installerbart media som källkod, men detta är inget som Red Hat hjälper till med, och sedan står du garanterat utan varje form av hjälp ifall fel uppstår.

har tillträde, på samma vis som för exempelvis Debian. Huvuddelen av de som arbetar på projektet är dock anställda på Red Hat, exempelvis hela styrgruppen. Det administrativa och finansiella huvudansvaret för Fedora Core har emellertid avsöndrats till en fristående stiftelse.

Från att ha varit ett projekt inriktat på strikta leveransdatum och marknadsföring övergick Red Hat Linux, omstöpt till Fedora Core, till att vara ett funktionalitetsdrivet projekt. Detta har uppenbarligen samband med att Red Hat flyttat fokus från sin grunddistribution till andra delar av verksamheten.

Fedora fungerar numera som en farmardistribution till Red Hat Enterprise Linux — Red Hat plockar med jämna mellanrum ut större delen av Fedora Core, lägger till och/eller tar bort saker och redistribuerar resultatet som Red Hat Enterprise Linux.

Om du vill installera det som en gång var Red Hat Linux för hemmabruk skall du installera Fedora Core. Denna distribution är inte lika stabil som Red Hat Enterprise Linux, är du ute efter maximal stabilitet, eller skall använda systemet i en större organisation, så bör du överväga ett inköp av Enterprise-produkten.

Med *Core* i namnet antyds att kärnan av paket som bygger upp Fedora Core har ett komplement: *Fedora Extras*, som tillhandahåller en stor mängd paket som inte anses vara absolut nödvändiga. Gränsen mellan de två delarna *Core* och *Extras* är något flytande och mellan två utgåvor av huvudsystemet *Core* kan paket flyttas från den ena delen till den andra. De delar som ingår i *Core* tenderar att vara paketerade av anställda på Red Hat, medan *Extras* innehåller många paket som i princip underhålls på frivillig basis.

4.3.1 Installation av Red Hat & Fedora Core

Red Hat profilerade sig tidigt mot att vara en installationsvänlig distribution. Detta resulterade i diverse halvgrafiska installationsprogram och sedermera det helt grafiska *Anaconda* som kan beskådas i figur 4.7.

Anaconda är så pass enkelt att använda att det inte torde kräva någon ytterligare kunskap eller beskrivning utöver det som redan nämnts i början av detta kapitel om bootstrap loader och partitionering. Om du vill ändra något i efterhand finns det grafiska verktyg som kan startas från operativsystemets ”Systemverktyg”-meny. Alla dessa ligger i */usr/sbin* eller */usr/bin* och tenderar att byta namn och utseende hela tiden, men heter oftast **system-config-foo** i Fedora Core.

En fördel som skiljer Anaconda från andra installationsprogram är att det skapar en så kallad *kickstartfil*. Detta är en receptbeskrivning av hur installationen gjordes, och kan sedan köras automatiskt om du öns-



Figur 4.7: Anaconda är ett fullgrafiskt installationsprogram som används av Red Hat och de flesta Red Hat-kopior.

kar installera flera datorer med exakt samma konfiguration, fast helt automatiskt. När du installerat ett Red Hat-system hittar du kickstartfilen i */root/anaconda-ks.cfg*.

Kickstartfilen är ganska enkel att förstå, och enkel att modifiera.[35] Om du vill kan du via kickstart lägga in extra paket som skall installeras ovanpå standardinstallationen, och du kan också sätta upp en filserver som lagrar alla de RPM-paket som används av installationen, plus några egna, om du vill. Poängen med det här är att en stor organisation ofta köper in en stor mängd likadana datorer som alla skall installeras på ett standardmässigt vis, även om var och en av datorerna skall ha en del inställningar som avviker från Red Hat Linux standard.

Eftersom användare i en större organisation oftast har sina hemkataloger lagrade på NFS-servrar och loggar in via NIS och liknande katalogtjänster,[36] kan en arbetsstation installeras om ifall det är något problem med den, eller om exempelvis hårddisken har kraschat eller skall bytas till en större. Då kan en administratör bara starta kickstartskriptet och sedan göra något annat medan Red Hat installerar sig självt. Vid manuell installation måste varje installation övervakas, och det tar en hel del arbetstid.

Kickstartfiler kan användas både för installation från CD-ROM och från nätverket. De används genom att kopiera filen till en vanlig diskett, och boota installationsskivan från bootprompten med kommandot:[37]

---

[35] Arbete pågår även på en grafisk kickstartkonfigurator, med namnet **system-config-kickstart** (tidgare **redhat-config-kickstart**).

[36] Se vidare avsnitt B.1 på sidan 418.

[37] För mer information om bootprompten i Linux, se sidan 184. För fullständig infor-

```
boot: linux ks=hd:fd0/ks.cfg
```

Huvuddelen av de inställningar som görs i Red Hat Linux sparas i */etc/sysconfig*. Till exempel säkerhetsnivåer, brandväggsinställningar, tidszon och språk, o.s.v. Detta sätt att lagra inställningar är inte standard på något vis och ingår heller inte i Filesystem Hierarchy Standard. (Se sidan 199.)

4.3.2 RPM-systemet

Red Hat (och sedermera Fedora) har det absolut mest välkända programpakethanteringssystemet: *RPM* vilket betyder RPM Package Manager, och alltså är en rekursiv akronym på samma vis som GNU. En förenklad version av RPM ingår också i standardiseringsinitiativet Linux Standard Base (se sidan 198).

RPM bygger på programpaket som normalt levereras med filändelsen *.rpm*, och som installeras i filsystemet. Mestadels hamnar de filer som finns i paketen i strukturen under */usr* men ofta nog även direkt i rotkatalogerna såsom */etc* och */bin*. En egen databas med information om installerade paket håller reda på vilka paket som är installerade samt vilka filer som hör ihop med vilka paket.[38] På så vis är det enkelt att lägga till och ta bort paket utan att skada andra installerade program.

Paketfilerna har följande egenskaper:

- De är kompilerade för en och endast en datorarkitektur. Red Hat stödjer ett fåtal arkitekturer, så huvudsakligen är det Intel 32-bitarsarkitekturen eller varianter av denna som avses. Ibland har Red Hat ändrat lite i programmen, så det är inte helt säkert att resultatet blir exakt detsamma som om du kompilerat och installerat programmet helt på egen hand.
- De innehåller ofta en del specialskript som körs före och efter det att själva filerna som ingår i paketet har installeras. Till skillnad från Debianpaketen är det ovanligt att RPM-paketen börjar ställa frågor till användaren, på sin höjd lämnas några meddelanden. Ett vanligt meddelande är att signaturen[39] inte kan verifieras, eller

mation om hur kickstart används, se Red Hats dokumentation, `http://www.redhat.com/docs/manuals/linux/` plocka fram din distributions *Customization guide*[27].

[38]Denna databas ligger i */var/lib/rpm*.

[39]RPM-paket kan signeras kryptografiskt med en digital signatur för att verifiera avsändaren, framför allt signeras alla paket från Red Hat själva.

att programmet kompilerats för en användare som inte finns på systemet (de skall i allmänhet kompileras som root).

Paketfilerna har namn i stil med *foo-2.0-12.i386.rpm* vilket skall utläsas som version 2.0 av paketet *foo*, där 12 är revisionen av själva paketet, för i386-arkitekturen.[40]

För att hantera manipulation, tillägg och borttagning av RPM-paket används följande kommandon:

rpm -i foo-1.0-1.i386.rpm *installerar* den första versionen av paketet *foo* för i386-arkitekturen i operativsystemet. Innan paketet installeras kontrollerar rpm ifall något annat paket måste installeras först, eller om paketet är beroende av att en annan version av något paket skall vara installerat. Då kan det bli problem och du tvingas i värsta fall installera en lång räcka olika paket, som alla måste anges i en lång rad efter **rpm -i**-kommandot.[41]

rpm -e foo *tar bort* (engelska: *erase*) paketet *foo* ur operativsystemet. Om något annat paket är beroende av att *foo* finns, kommer detta inte att tillåtas utan vidare. För en applikation på hög nivå är det dock för det mesta okomplicerat.

rpm -U foo-2.1-1.i386.rpm *uppdaterar* paketet *foo* till en nyare version. Att bara skriva **rpm -i foo-2.1-1.i386.rpm** ifall en äldre version av *foo* är installerat kommer *inte* att fungera. Du måste ange explicit att det är fråga om en uppgradering.

rpm -q foo *frågar* (engelska: *query*) om information rörande vilken version av paketet *foo* som är installerat.

rpm -qf /usr/bin/foo tar reda på vilket paket den där filen */usr/bin/foo* kom ifrån.

rpm -qi foo visar fullständig information om paketet *foo*.

rpm -ql foo visar en fullständig lista över filer som kom med paketet *foo*, och vart de har installerats.

rpm -qa visar en fullständig lista över alla paket installerade i ditt system. Även globbningsvarianter som **rpm -qa foo*** fungerar bra.

Tidigare användes kommandot rpm även av distributörer för att *bygga* paket, men denna funktionalitet har numera separerats ut till det separata kommandot **rpmbuild**. För ytterligare information om RPM-systemet, se `http://www.rpm.org/`.

[40]Se vidare avsnitt 3.5, sidan 126 om versionsnummer.

[41]Du *kan* förvisso skriva **rpm -i -nodeps foo-1.0-1.i386.rpm** men det rekommenderas *inte*. Gör bara detta om du är absolut säker på vad du sysslar med.

4.3.3 Up2date och YUM

Huvudtanken med det ursprungliga Red Hat Linux var att uppdateringar av operativsystemet skulle ske i form av nya versioner av hela distributionen, och inte kontinuerligt. I begynnelsen fick säkerhetsuppdateringar därför laddas ned från en webbsida och installeras manuellt med **rpm**-kommandot. I längden blev det ohållbart att kunderna varje dag skulle besöka en webbsida och sedan manuellt installera uppdateringar på alla datorer. För att lösa detta uppfanns up2date-funktionen, som på varje dator kontinuerligt kontrollerar om uppdateringar finns tillgängliga, och gör användare uppmärksamma på om dessa behöver installeras. Detta sker i form av en liten ikon på varje dators skrivbord.

Up2date är dock inget perfekt verktyg: det är oklart om avsikten är att alla användare skall kunna köra det hur som helst, bara root kan ju installera uppgraderingarna, så varför skall alla användare göras uppmärksamma på något de inte kan åtgärda?

I Fedora Core finns ytterligare ett installationsverktyg med namnet *YUM*, av *Yellow Dog Updater, Modified*, vilket betyder just att det är uppdateringsprogrammet från Yellow Dog Linux i en modifierad variant. Det är liksom APT ett beroendehanteringsverktyg som kan installera ett paket och alla dess beroenden vid begäran. Om du installerat Fedora Core är det YUM du i första hand skall använda för att installera nya paket utöver de som finns på grundsystemets CD-skivor, och för att uppdatera dina befintliga installerade paket.[42] YUM används vanligen manuellt som root med:

```
yum update
```

vilket uppgraderar de paket som eventuellt kommit i nya utgåvor. Det **apt-get** är för Debian är YUM för Fedora Core.[43] **yum install foo** har t.ex. den förväntade effekten att installera paketet *foo* och med **yum list "*foo*"** listar alla paket från alla tillgängliga paketkällor med namn som innehåller strängen *foo*. **yum remove foo** tar bort ett paket men kan lika gärna ersättas med **rpm -e foo**.

Något bra grafiskt verktyg i stil med Debians **synaptic** finns i dagsläget inte, och för att ställa in paketkällor för YUM krävs i princip att du manuellt går in och editerar filen */etc/yum.conf* eller lägger till en fil i

[42] I stora organisationer använder många även Novells verktyg *ZENworks Linux Management*, med ursprung i företaget Ximians verktyg *Red Carpet* för att massuppdatera stora mängder datorer. Detta system sköter även sådana administrativa uppgifter som registrering av anläggningstillgångar i form av datorer (engelska: *asset management*).

[43] **apt-get** finns även portat till Fedora Core och RPM-systemet och användes mycket under en period då inget annat alternativ fanns. Den ökande betoningen av YUM från Red Hats sida har dock gjort detta till den dominerande metoden.

katalogen */etc/yum.repos.d* som innehåller de paketkällor du vill använda. Detta är dessutom något som en snittanvändare ofta blir tvungen att göra: tillgången till paket från Fedora Extras är från och med Fedora Core 4 påslagen per automatik, men ofta är någon oberoende paketkälla som Livna[44] eller FreshRPMS[45] nödvändig för att installera mediaspelare, MP3-stöd, Java, drivrutiner för exotiska grafikkort och annat som av principskäl inte ingår i Fedora Core. Den inofficiella FAQ:en på `http://www.fedorafaq.org/` tillhandahåller en egendefinierade inställningsfiler och ger en viss vägledning i detta ofta helt nödvändiga moment.

4.3.4 Felrapportering

Om du hittar ett fel i Red Hat Linux eller Fedora Core så bör detta rapporteras tillbaka till projektet. Problemrapporterna skall vara precisa, härledda till exakt ett paket, och innehålla en beskrivning av hur felet kan reproduceras. Du hittar Red Hats felrapporteringssystem på `http://bugzilla.redhat.com/`. Detta system används även av Fedora Core.

4.4 Linux From Scratch

Officiell hemsida:
`http://www.linuxfromscratch.org/`

Vissa människor bor i elementhus, d.v.s. hus byggda av standardkomponenter. De flesta av oss bor i ett hus som vi inte själva har byggt eller varit med om att göra ritningar till. Men en del av oss är individualister som faktiskt bygger våra egna hus. Ja, vi kanske tar hjälp av en firma för att gjuta grunden, installera elektriciteten och kapa plank, men i grund och botten formger och bestyckar vi kanske våra egna hus. Beundrar vi sådana människor? Kanske. Tycker vi att de besitter någon överjordisk kunskap, för den skull att de lyckas bygga sitt eget hus? Nej, inte precis. Vi kan alla föreställa oss att även om det förmodligen tar en förskräcklig tid och kostar mer än ett standardhus, så går det visst att lära sig sådana saker som fordras under tiden som bygget pågår.

Således borde vi inte bli förvånade om somliga människor bygger sina egna operativsystem. Men någon form av vägledning behövs kanske, lite ritningsförslag, tips om hur andra har gjort, vilka saker som måste finnas för att en del basala ting skall fungera. Den distribution

[44] Se: `http://www.livna.org/`
[45] Se `http://www.freshrpms.net/`

som kallas *Linux From Scratch* erbjuder just detta. Det engelska uttrycket *to do something from scratch* betyder ungefär *att göra något från grunden upp*.

Linux From Scratch är en bok — inga disketter, ingen CD-ROM, inga ISO-filer, ingenting annat. Författaren är huvudsakligen Gerard Beekmans. Det är rent faktiskt ett paket med HTML-filer (eller råtext eller liknande) som du laddar ner och läser dig igenom. De ger en steg-för-steg-beskrivning av hur GNU/Linux byggs upp från små atomer (som **bash** och **gcc**) till allt större molekyler, tills en hel, självständig GNU/Linux-installation till sist har skapats på din dator. Boken är på engelska och därför fordras det att du är hyfsat bra på att läsa detta anglosaxiska språk.

Det finns goda skäl att installera Linux From Scratch. Vill du bli en riktig expert på hur GNU/Linux fungerar är detta ett av de bättre sätten, kanske det bästa. Vill du bygga Linux för inbyggda system eller för ett annat ändamål som kräver att det skall vara extremt välanpassat och litet, till exempel en *X*-terminal, så är Linux From Scratch den perfekta guiden till att första gången prova på allt det som du behöver kunna. Efter att du installerat Linux From Scratch har du en miljö som liknar den som utvecklarna av Linuxkärnan i allmänhet själva använder.

Filosofin bakom boken är att du skall veta vad varje paket som ingår i distributionen är till för, och exakt varför det behövs. Därför förklaras det lite översiktligt vad varje paket har för funktion innan du får instruktioner som talar om hur det skall installeras.

Linux From Scratch förutsätter att du redan har någon annan installation körande på din dator. Vilken installation som helst som innehåller **gcc**, **make**, och liknande programmeringsverktyg duger. Det krävs utrymme för att skapa en ny partition på din hårddisk där det nya operativsystemet skall ligga, c:a 1 gigabyte, så återgå gärna till avsnitt 4.1.4 för att repetera något om hur detta går till.

I övrigt är det mesta i denna installation upp till dig själv. Linux From Scratch har gjort en del programlappar (patchar) för olika program som behövs för installationen, och dessa finns också hänvisade till i boken, så så länge du följer steg-för-steg-anvisningen borde inga större problem uppstå. Uppstår problem eller frågor i alla fall, så finns det flera olika epostlistor knutna till Linux From Scratch som du kan få hjälp ifrån.

När det gäller mer avancerade högnivåprogram som fönstersystemet *X* och GNOME så ger Linux From Scratch ingen särskild hjälp med dessa. Istället finns det ytterligare en bok: *Beyond Linux From Scratch* som beskriver hur du installerar *X*, en hoper grafikbibliotek, GNOME och/eller KDE, texteditorer o.s.v. Den som verkligen gått igenom he-

la denna procedur får snart känslan av att det kanske inte kan vara så svårt att sätta ihop sin egen GNU/Linux-distribution.

4.5 Alla de övriga

Det lär finnas ungefär 1000 olika GNU/Linux-distributioner. Det finns ingen möjlighet att ta upp alla här, men jag passar på att nämna några hyfsat populära varianter och hur de relaterar till de två föregående:

SuSE Linux skapades 1996 av konsultföretaget *Software- und System-Entwicklung*, som i sin tur grundades av bland andra Roland Dyroff runt 1992. SuSE bestod i början av en mängd programlappar för SLS Linux för att bland annat hantera tyska språket ordentligt. Företaget var offensivt på USA-marknaden, och lyckades etablera ett samarbete med IBM innan det såldes i sin helhet till Novell i november 2003. Novell hade då redan tidigare köpt upp Ximian och positionerar sig därmed som en av de kanske viktigaste aktörerna inför framtiden. Specifika egenheter med SuSE är det egna installationsprogrammet *YaST* (Yet another Setup Tool) som är ungefär likvärdigt med Red Hats Anaconda. SuSE använder Red Hats paketsystem RPM. Liksom Red Hat Enterprise Linux måste SuSE Linux köpas, det kan inte laddas ned på Internet, äldre versioner kan installeras via FTP. Novell har dock aviserat att en fritt nedladdningsbar variant skall skapas under namnet *openSUSE* i analogi med hur Fedora Core fungerar för Red Hat.[46]

Yellow Dog Linux kallas bland användare för ”Red Hat för PowerPC” och skapades år 1999 av Coloradoföretaget Terra Soft Solutions. Denna version av GNU/Linux går lätt att installera på alla Macintosh-datorer med en PowerPC-processor, och är av den enkla anledningen en av de snabbaste GNU/Linux-distributioner som finns att tillgå. Förutsatt att den körs på en splitterny Mac är den i dagsläget förmodligen den snabbaste Linux som kan fås för rimliga summor pengar. Yellow Dog Linux använder också den Red Hats RPM-system och är naturligtvis också anfader till beroendehanteringsverktyget YUM som ingår i Fedora Core.

Mandriva Linux är en Red Hat-baserad distribution ursprungligen skapad av Gaël Duval i Caen, Frankrike i juli 1998 på basen av Red Hat Linux 5.1, och kallades därför med konsekvens för Mandrake Linux 5.1. (Namnet Mandrake fick överges år 2004 eftersom detta var ett varumärke som ägdes av serietidnings-holdingföretaget

[46]Se: `http://www.opensuse.org/`

King Features.) I princip var Mandrake då bara en Red Hat-installation plus skrivbordsmiljön KDE, som Red Hat då vägrade använda. Liksom Red Hat använder det RPM-systemet, och i December 1998 grundades företaget MandrakeSoft för att underhålla och distribuera Mandrake Linux. En egenhet med Mandrake är att det är kompilerat exklusivt för att fungera på Intel Pentium-processorer och senare, det är alltså mycket profilerat mot skrivbordsanvändare med vanliga IBM PC-kloner. Ett pakethanteringssystem kallat **urpmi** har i Mandriva Linux ungefär samma egenskaper som Debians **apt-get** eller Fedora Cores **yum**.

Gentoo Linux skall vara ovanligt populärt i Sverige. Denna distribution har en totalt annorlunda inställning och profil än andra: Gentoo levereras i princip okompilerat. Under installationen kompilerar Gentoo i allmänhet självt alla programpaket som behövs för att köra det, och det optimeras då samtidigt för den dator där det installeras. Om du har erfarenhet av att kompilera saker inser du kanske att det därför inte går särskilt snabbt att installera Gentoo, i synnerhet inte på en äldre maskin, och av den anledningen finns numera även möjligheten att använda vissa förkompilerade paket.

För att klara av allt kompilerande använder Gentoo ett eget pakethanteringssystem kallat *portage* som är baserat på det portssystem som används av BSD-projekten, och som alltså bara innehåller okompilerad källkod och en fil som ställer in och startar kompileringen. Gentoo stödjer flera olika datorarkitekturer, däribland Sun SPARC och PowerPC. Gentoo är förmodligen en bra kompromiss om du vill ha hög grad av kontroll över ditt system, men ändå inte vill kämpa dig igenom allt handarbete som krävs av Linux From Scratch, men som ändå kan tillräckligt mycket om GNU/Linux-system för att ändå skriva ordentligt med skalkommandon för att komma igång. Jag har själv använt det till exempel för att bygga ett system fullständigt optimerat för en ren Pentium-MMX-dator, vilket ju inte är så vanligt.[47]

Gentoo Linux lämpar sig även speciellt för den som tycker att pakethanteringssystem som RPM och/eller DEB är rena skräpet och ställer till mer problem än de löser.

Linspire (tidigare Lindows OS) är inriktat på så hög grad av ”Windows-likhet” som möjligt, för att göra ett byte lätt för den som bara använt Windows tidigare. Liksom Red Hat Enterprise Linux eller

[47] D.v.s. GCC kompilatorflagga -march=pentium-mmx, och arkitekturbeteckning i586.

SuSE Linux måste Lindows köpas, och kan inte laddas ner från Internet.

Knoppix eller **Gnoppix** är två distributioner baserade på Debian, som dock är avsedda att köras direkt från en CD, utan att installera operativsystemet på hårddisken. Om du laddar ned och kör Knoppix eller Gnoppix kan du testa om det verkar som att GNU/Linux skulle kunna fungera bra på din dator. Detta rekommenderas dock inte om din dator har ett litet internminne — 256 megabyte eller mer och en snabb dator krävs för att det skall vara någon mening med detta, annars kommer systemet att vara outhärdligt långsamt.

Idén med att kunna starta GNU/Linux direkt från CD är verkligen inte ny, Yggdrasil Linux kunde göra detta redan år 1993, men distributioner av detta slag har nu åter blivit populära. Även SuSE Linux finns i en version som kan köras direkt från CD, vilket är avsett för att användas som test, innan du beslutar om du skall köpa det ”riktiga” SuSE.

En extremare variant är **LNX-BBC** (av engelskans *Bootable Business Card*) som är speciellt framtagen för att passa på en vistkorts-CD (en väldigt liten CD) och som är till för att plocka fram ur plånboken när du behöver komma in i en maskin som befinner sig i ett osunt läge, ungefär som en bootdiskett.

Som du kan se har vi SuSE Linux från Tyskland och Mandriva Linux från Frankrike, vilket har gett dessa två distributioner ett speciellt gott stöd för tyska respektive franska. (SuSE har en filial i Tjeckien och därför även bra stöd för tjeckiska.) Denna lokaliseringstendens är inte unik för SuSE och Mandrake: Connectiva Linux: bra spanska, Red Flag Linux och Turbolinux: bra kinesiska, ASPLinux: bra ryska och slaviska. Fler exempel saknas säkert inte, och mängden distributioner fluktuerar varje dag.

4.6 Program utanför distributionen

Distributionerna tillhandahåller alltså färdigkompilerade, och ibland även hyfsat testade programpaket som sätts samman till ett helt operativsystem.

Vad händer då den dag då ett program inte finns som ett färdigt paket till ditt operativsystem, eller om du plötsligt vill ha en nyare version av ett program än det som följde med distributionen?

Första alternativet är att titta efter *inofficiella paket*. Detta innebär att du börjar leta efter paket som tillverkats av någon som inte är involverad i respektive distribution. För Red Hat finns stora mängder paket som inte alls framställts av Red Hat, och som lätt kan laddas ned och installeras. En populär nedladdningssajt är `http://freshrpms.net/` och via `http://www.rpmfind.net/` kan du hitta många lustiga paket. Till Debian är utbudet inte lika stort, en samlingspunkt för inofficiella Debian-paket finns på `http://www.apt-get.org/`. Det finns inget som garanterar äkthet och kvalitet på dessa paket.

Andra alternativet är att söka efter kompatibla paket. Ett RPM-paket gjort för en Mandrake-installation kan heta exempelvis *foo-1.2-13mdk.rpm* och är väl förvisso bara garanterat att fungera med Mandrake, men oddsen är ändå goda för att det fungerar lika bra med Red Hat. Det finns till och med ett program som heter **alien** och som kan konvertera mellan DEB och RPM-paket, så att Debian-användare kan använda RPM:er och vice versa. Det kan även hantera Stampede Linux SLP-paket och de enkla *.tgz*-filer[48] som används i Slackware Linux. Denna metod fungerar väl sisådär. Det finns viss risk för att du kladdar till ditt system rejält på detta vis.

Tredje alternativet är att installera en förkompilerad binär fil, som inte ingår i något paket alls. En del proprietära program till GNU/ Linux brukar levereras enbart på detta vis, exempelvis RealPlayer och Macromedia Flash. Lägg sådana program direkt i katalogen */usr/local/bin* om du får möjlighet att välja vart de skall placeras. Ibland vill dessa program rent av installera sig i din hemkatalog. För denna typ av program saknas en etablerad standard, och alla leverantörer gör på sitt eget vis.

Det sista alternativet, som varje GNU/Linux-användare förr eller senare kommer att begagna sig av, är att kompilera program själv, med utgångspunkt från källkodsarkiv. Detta ganska komplicerade tillvägagångssätt avhandlas i kapitel 8 på sidan 283. Detta alternativ är ofta det bästa om du inte kan få tag på ett paket för just din distribution.

4.7 GNU/Linux för inbyggda system

När det gäller inbyggda system — och med detta menas en digitalteknisk apparat som utför en viss uppgift, såsom en mobiltelefon eller ett kylskåp — har GNU/Linux kommit att spela en allt större roll. Att installera och utveckla under GNU/Linux på ett inbyggt system är na-

[48] En *.tgz*-fil är en gnuzippad *.tar*-fil, dvs detsamma som en *.tar.gz*-fil, bara med ett förkortat namn.

turligtvis inget för den vanliga användaren, och vi skall här bara göra en snabb orientering för att skaffa oss en förståelse för hur detta går till. Om du inte är intresserad av inbyggda system eller upplever detta avsnitt som obegripligt kan du hoppa över det.

Inbyggda system har speciella egenskaper: operativsystemet skall vara så litet som möjligt, d.v.s. kräva så lite minne och hårddiskutrymme som möjligt.

Vidare behöver alla som utvecklar inbyggda system en såkallad *verktygskedja* (av engelskans *toolchain*) vilken utgörs av de kompilatorer, avlusare och andra specialverktyg som krävs för att utföra utveckligsarbete på ett visst inbyggt system. Denna verktygskedja innehåller i allmänhet:

- Kablar, anslutningsdon och annan elektronik som behövs för att ansluta ditt inbyggda system till en vanlig PC. Vissa inbyggda system använder en mycket okomplicerad serieport till detta, andra använder USB, men det vanligaste idag är att använda *JTAG*[49] vilket är ett standardgränssnitt som tillhandahåller en speciell serieport med fyra eller fem pinnar, vilken via ett standardiserat lågnivåprotokoll kan ta rent elektronisk kontroll över systemet.

- Mjukvara som kan kommunicera med systemet via t.ex. JTAG. Det finns ett flertal JTAG-orienterade projekt för GNU/Linux, beroende på vad det skall användas till. Vissa system använder även en s.k. hårdvarudebugger, vilket kräver ytterligare anslutningsdon och gör det möjligt att styra processorn i det inbyggda systemet fram och tillbaka i exekveringsflödet, läsa av minnesinnehållet och hur det förändras vid körning av vissa kodsegment o.s.v. Detta är ofta en helt nödvändig (och dyrbar, samt proprietär) utrustning.

- En kompilator och länkare (i allmänhet någon variant av GNU Compiler Collection) som kör på din vanliga PC, men som inte kompilerar kod till den processor och det C-bibliotek och det utfilformat som en vanlig PC använder, utan som matar ut något som ditt inbyggda system kan använda sig av. Detta har i allmänhet varit ganska jobbigt att åstadkomma, men för nya projekt brukar det gå bra att använda *crosstool*, ett skript som skrivits av Dan Kegel och som hjälper till med alla dessa detaljer.[50]

[49]Joint Test Action Group, IEEE-standard 1149.1 utvecklades ursprungligen för att testa högskaligt integrerade kretsar, men används numera för allt möjligt som har med utveckling av inbyggda system att göra.

[50]Se: `http://www.kegel.com/crosstool/`

När du väl har din verktygskedja på plats kan du börja plocka de bitar som behövs för ditt inbyggda system. Processen påminner om att bygga upp *Linux From Scratch* (se sidan 162) fast i ännu något mindre skala.

För inbyggda system krävs i allmänhet en anpassad (mycket liten) Linuxkärna, vilket betyder att du måste konfigurera samt kompilera den själv. Du får dessutom själv skriva drivrutiner för all speciell hårdvara som ditt inbyggda system kan tänkas använda. Om processorn i ditt inbyggda system har en MMU (av engelskans Memory Management Unit, minneshanteringsenhet) kan du använda den helt vanliga standardkärna som finns på *kernel.org*. Har din processor inte en minneshanteringsenhet får du använda dig av den modifierade kärna som μCLinux (utläses micro-controller Linux) tillhandahåller.[51]. Denna är sedan version 2.6 av Linuxkärnan integrerad i standardkärnan, men projektet tillhandahåller ännu stora mängder tilläggskod och specialverktyg för att använda Linux på inbyggda system, vilket gör att även den som använder en processor med MMU bör vara bekant med detta projekt.

Utöver kärnan behövs naturligtvis ett C-bibliotek. Det som brukar användas är uClibc (vilket utläses micro-controller C library).

När väl dessa komponenter har kommit på plats vill du säkert prova att öppna någon form av skal i ditt nya operativsystem. Det som brukar användas är GNU-projektets Bourne-Again Shell (**bash**), kompilerat från källkod. För övriga nödvändiga verktyg såsom **ls**, **chmod** o.s.v. brukar ett stort grötpaket innehållande alla vanliga kommandon i nedskalade versioner användas. Detta går under namnet *BusyBox*.[52] BusyBox har byggts genom att ta källkod från BSD och GNU-projektet och komprimera och prioritera innehållet så att bara det absolut nödvändigaste finns med.

Filsystemet i ett inbyggt system brukar vara av romfs-typ. Det innebär att det är extremt kompakt och enbart avsett för läsning, så innehållet kan inte modifieras. Detta gör för det mesta ingenting: när filsystemets innehåll skall bytas måste i allmänhet hela mjukvaran i systemet bytas, såväl operativsystem som filsystem, och då utgör dessa saker tillsammans en s.k. *firmware* som måste bytas i ett stycke. Om delar av filsystemet skall vara skrivbara, t.ex. i form av loggar och liknande, kan de ju monteras ovanpå det befintliga, enbart läsbara filsystemet.

Om allt detta verkar jobbigt kan det kanske vara en tröst att det säljs kompletta s.k. *labbkort*, vilket är ett kretskort med en processor, lite minne och (vanligen) en JTAG-port och kanske en display, som är anpas-

[51]Se: `http://www.uclinux.org/`

[52]Se: `http://busybox.net/`

sat för att utgöra grunden till ett inbyggt system. Labbkorten levereras med en viss Linuxkärna, ett C-bibliotek och en komplett verktygskedja. Tanken med labbkorten är att det som finns på dem sedan kan krympas ned till storleken på det (oftast mycket mindre) inbyggda system där det skall användas. Labbkorten köps därför ofta tillsammans med kretsscheman och komponentlistor så att det är lätt att bygga små system som fungerar precis som labbkorten. Labbkort av det här slaget tillhandahålls vanligen av processortillverkarna, t.ex. ARM, IBM eller svenska Axis.

Flera inbyggda system kräver realtidsrespons, vilket i tekniska krav motsvaras av preemptiv multikörning.[53] Linux har på senare tid fått ett ordentligt stöd för preemptiva trådar och processer, och det finns därför många användare av äldre realtidsoperativsystem som börjat titta på att migrera sina befintliga lösningar till GNU/Linux. Ett företag som har detta som specialitet är MontaVista som inriktat stora delar av sin verksamhet på att skapa kompatibilitetslager mellan GNU/Linux och andra realtidsoperativsystem såsom Wind River Systems *VxWorks*.

[53]Detta innebär att kärnan kan avbryta mindre viktiga aktiviteter — och därtill räknas även mindre viktiga aktiviteter som utförs av kärnan själv — till förmån för viktiga aktiviteter, t.ex. en signal i en pacemaker eller dylikt som måste prioriteras framför andra arbetsuppgifter.

KAPITEL 5

GNU/Linux-projekten

Det här kapitlet avhandlar de tre största och viktigaste projekten som tillhandahåller den mjukvara som bygger upp ett GNU/Linux-system: GNU, Linux och BSD. Av dessa är GNU och Linux så viktiga, att de fått ge namn åt operativsystemet som helhet. Dessa två är vad som krävs för att få igång någonting som är tillräckligt avancerat för att du skall kunna starta ett POSIX-kompatibelt skal och köra dina mest grundläggande kommandon.

5.1 GNU-projektet

Officiell hemsida: `http://www.gnu.org/`

GNU-projektet, vilket utläses *GNU's Not Unix*, annonserades den 27 september 1983 och startades formellt i januari 1984 med målet att bygga ett fullständigt, fritt, UNIX-kompatibelt operativsystem. Den ursprungliga annonsen från Usenet-gruppen net.unix-wizards börjar:[1]

> Fri Unix!
>
> Med början denna Thanksgiving kommer jag att skriva ett fullständigt Unix-kompatibelt mjukvarusystem kallat GNU

[1] Min översättning

> (betyder Gnu's Not Unix), och ge bort det fritt[2] till var och en som kan använda det. Tillskott av tid, pengar, program och utrustning behövs i stor grad.
>
> Till att börja med kommer GNU att vara en kärna plus alla verktyg som behövs för att skriva och köra C-program: texteditor, skal, C-kompilator, länkare, assemblator och några andra saker. Efter detta kommer vi att lägga till en textformaterare, en YACC, ett Empire-spel, ett kalkylark, och hundratals andra saker. Vi hoppas i slutänden kunna tillhandahålla allt användbart som i vanliga fall kommer med ett Unix-system, och allt annat användbart, inklusive online och tryckt dokumentation.[32]

Enmansorganisationen bakom detta tillkännagivande var Richard M. Stallman, bland vänner kallad RMS. Denne hade bakom sig en längre fejd med LISP-maskintillverkaren Symbolics som hade anställt alla hans kollegor från laboratoriet för artificiell intelligens vid Massachusetts Institute of Technology (MIT). Stallman uppfattade detta som en aggresiv handling mot den arbetskultur som frodades där, och ville därför skapa en fristad för programmerare som jobbade med datorprogram med utgångspunkt från att de skall vara fritt tillgängliga, modifierbara och användbara.[3]

Det första GNU-programmet som skrevs var Bison, en ersättning för det något obskyra programmet YACC, vilket står för *Yet Another Compiler Compiler*, och är ett verktyg för att skapa kompilatorer för olika programspråk. Bison förbättrades av Stallman för att kunna hantera kompilatorbeskrivningar gjorda för YACC. Bison var liksom flera andra program i det tidiga GNU donerade av ursprungsförfattaren, som varit positivt inställd till projektet.

Under sitt arbete med sina nästa två projekt, GNU C Compiler och GNU EMACS, började Stallman sälja kopior av dessa mjukvaror på band för några hundra dollar styck för att på så vis kunna betala mat och hyra för sig själv och projektet. En arbetsplats för honom själv tillhandahölls av MIT.

År 1985 instiftade Stallman tillsammans med några vänner organisationen *Free Software Foundation*, som fick i uppdrag att förvalta och

---

[2] Stallman använder det engelska, och i dessa sammanhang så omtvistade ordet *free*, och markerar här inte särskilt tydligt att det handlar om frihet och inte om pengar, därav kan slutsatsen dras att denna skillnad i begynnelsen kanske inte var särskilt tydlig ens för Richard Stallman, utan var något som utvecklades efter hand.

[3] Den psykologiska grunden till varför Stallman startade detta projekt utreds närmare i biografin *Free As In Freedom*[39], arbetskulturen vid MIT-laboratoriet beskrivs utförligt i boken *Hackers — Heroes of the Computer Revolution*[17].

utveckla GNU-projektet. En mängd program började att utvecklas inom GNU, de två viktigaste är med största sannolikhet *GNU C Compiler* (senare omdöpt till *GNU Compiler Collection* men oftast kallad bara *GCC* efter hur den används i skalet med kommandot **gcc**) och GNU C Library, ibland kallat *glibc*. Dessa två komponenter gjorde den fortsatta utvecklingen av projektet möjlig, eftersom nästan alla GNU-program är skrivna i programspråket C. För att utveckla C-kompilatorn bad Stallman Andrew Tannenbaum om lov att få använda hans C-kompilator *Amsterdam Compiler Kit*. Detta föll inte väl ut, och han beslutade därför att skriva sin egen kompilator, vilket gick med raketfart. GCC blev snart internationellt ryktbar.

Samma år uppstod en konflikt med den som senare kom att bli upphovsmannen till programspråket Java, James Gosling. De tidiga versionerna av GNU EMACS hade nämligen baserats på en fri version kallad Gosling Emacs, vilken Stallman fått ett muntligt löfte från Gosling om att få använda i GNU-projektet, och nu hade Gosling sålt sin implementation till företaget UniPress, vilka snabbt förbjöd Free Software Foundation att sprida GNU EMACS. Detta gjorde att stora delar av GNU EMACS fick skrivas om från början. Denna och andra incidenter ledde till författandet av GNU General Public License, vilken beskrivs utförligare i avsnittet om licenser på sidan 120.

Sedan dess har Free Software Foundation inom ramen för sitt GNU-projekt successivt utvecklat och uppdaterat alla de verktyg och program som återfinns i figur 5.1, och därutöver en hel rad andra, till exempel GNU EMACS, Autoconf, Automake, GNU Make o.s.v. De viktigaste programmen är samlade i källkodspaket[4], exempelvis innehåller programpaketet *GNU Fileutils* kommandon som **ls**, **chmod** o.s.v. som är helt oumbärliga för att använda ett POSIX-system.

Det ojämförligt viktigaste av dessa källkodspaket är *GNU C Library*, ofta kallat *glibc*, vilket är ett programbibliotek skrivet av huvudsakligen Roland McGrath och Richard Stallman och som används av i princip *alla* Linuxprogram. Om du installerar hela Linux från källkod (som med Linux From Scratch-distributionen) är *glibc* normalt det allra första du installerar för att åstadkomma ett självständigt system. Om du aldrig programmerat själv kan vikten av *glibc* kanske vara svår att förstå, nöj dig i så fall med att konstatera att det är *mycket* viktigt. Låt dig heller inte förledas av namnet ”C Library”, ty *glibc* används i praktiken av alla programspråk, eftersom andra programspråks kompilatorer och standardbibliotek i allmänhet är skrivna i programspråket C, och därför i slutänden ändå använder *glibc*. Även stora delar av Bourne Again Shell

[4] I praktiken är detta ett filarkiv, ofta något i stil med *foo.tar.gz*, det är *inte* fråga om samma paketbegrepp som för distributioner.

Kommandon från GNU-projektet

alias†	ar	awk	basename	bc	bg†
break†	c99 (gcc)	cat	cd†	chgrp	chmod
chown	cksum	comm	command†	continue†	cp
csplit	cut	date	dd	df	diff
dirname	du	echo	env	eval†	exec†
exit†	expand	export†	expr	false	fc†
fg†	find	fold	gencat	getconf	getopts†
grep	hash†	head	iconv‡	id	jobs†
join	link	ln	locale‡	localedef‡	logname
ls	m4	make	mkdir	mkfifo	mv
nice	nl	nm	nohup	od	paste
patch	pathchk	pr	printf	pwd	read†
readonly†	renice‡	return†	rm	rmdir	sed
set†	sh†	shift†	sleep	sort	split
strings	strip	stty	tail	tee	test
time	times†	touch	tr	trap†	true
tsort	tty	type†	ulimit†	umask†	unalias†
uname	unexpand	uniq	unlink	unset†	wait†
wc	who	xargs	yacc (bison)	zcat	

Kommandon från Linuxutvecklare

at	batch	fuser	ipcrm	ipcs	mesg
newgrp	ps	write			

Kommandon från BSD-projektet

cal	crontab	ed	file	kill	logger
man	more	pax	renice	talk	tput
yacc					

Kommandon från andra projekt

ctags	ex (vim)	lex/flex	lp	vi (vim)

Figur 5.1: POSIX-verktyg (kommandon) som utvecklats av de olika delprojekten i en vanlig distribution (Red Hat Linux 9). † = ingår som del i Bourne Again Shell, **bash**, ‡ = ingår som del i GNU libc.

och alla andra skal är beroende av anrop till färdiga konstruktioner i *glibc*.

Utöver *glibc* är ditt system i allmänhet beroende av alla GNU-projektets programmeringsverktyg för att över huvud taget bli till. GNU C Compiler (**gcc**), Binutils, Fileutils, Textutils, Sh-utils, Gettext, Tar, Grep, Gzip och dess programbibliotek Zlib, Make, Autoconf och Automake är alla källkodspaket som få programutvecklare klarar sig utan. De installeras också ofta tillsammans med din distribution eftersom de kan vara bra att ha, i synnerhet om du vill kompilera saker själv (se kapitel 8).

Utöver detta har vi redan nämnt GNU GRUB och GNU Parted som är oumbärliga vid installation, och som tidigare nämnts föredrar GNU-projektet också ofta att dokumentera sina program i *texinfo* istället för som *man*-sidor.[5]

År 1990 hade projektet redan färdigställt större delen av det operativsystem som skulle få namnet GNU OS — det slutgiltiga operativsystemet GNU. Emellertid hade kärnan *HURD* (se sidan 23) kraftigt försenats. Svårigheterna med att skriva en mikrokärna hade grovt underskattats, och fördelarna med metodiken hade överskattats. Stallman hade då ännu en gång kontaktat Andrew Tanenbaum i syfte att få honom att dela med sig av kärnan till det operativsystem med namnet *Minix* som denne skrivit, och som skickades med hans böcker om operativsystemteori. Tanenbaum förhöll sig även denna gång kallsinnig till Stallmans förfrågan.

5.2 Linuxprojektet

Officiell hemsida: `http://www.kernel.org/`

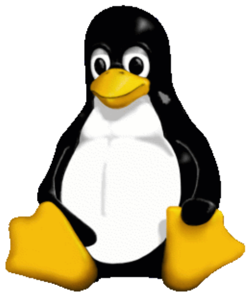
Talk is cheap. Show me the code.
— Linus Torvalds[6]

Det var i detta läge som Linus Torvalds, född 1969, började fundera på operativsystem. Denne hade börjat studera vid universitetet i Helsingfors 1988, och hade där mött UNIX i form av Digitals UNIX-variant Ultrix.[7]

[5]Denna policy har dock lagts om något: det nya XML-baserade dokumentationsformatet DocBook skall framledes användas, men kan vid behov exporteras till såväl texinfo som man-sidor. Förändringen kommer att ske så fort projektet funnit ett bra sätt att konvertera sina befintliga texinfo-filer till DocBook.

[6]Från Linux Kernel Mailing List.

[7]Författaren delar förnamn, modersmål och första POSIX-operativsystemupplevelse med Torvalds. Några andra likheter existerar vad jag vet inte, Torvalds har exempelvis namnet efter Linus Pauling, medan mitt kommer från en bröllopsannons.



Figur 5.2: Pingvinen Tux valdes som logotyp för Linux i och med introduktionen av Linux 2.0. Logotypen ritades av Larry Ewing.

Torvalds första dator hade varit en Commodore VIC-20, därefter en trotjänande Sinclair QL, men inför studierna i Helsingfors behövde han en ordentlig dator, så i januari 1991 köpte Torvalds en ny IBM-kompatibel PC.[8] Den nya datorn använde han till en början mest för att spela *Prince of Persia* i Microsoft DOS.

Med hjälp av denna maskin skrev han en del program i assembler för att testa processbytesmekanismen som fanns inbyggd i Intel 80386, och som gjorde det möjligt att programmera ett mer avancerat operativsystem än DOS på denna maskin. I mars fick han en kopia av Tanenbaums operativsystem Minix som han genast installerade och började programmera med. Ett av de första programmen han installerade var GNU C Compiler version 1.37.

Med kompilatorn implementerade han återigen sitt processbytessystem. I april fick han behov av en terminalemulator för Minix för att kunna ringa upp universitetet via modem och läsa nyhetsgrupper på Internet, framför allt *comp.os.minix*, den plats där Minix diskuterades. Han skrev då om processbytessystemet till en terminalemulator med två processer: en som läste från modemet och en som samtidigt skrev till det. Lite senare lade han även till diskhantering, eftersom han ville kunna ladda ner nya program. Han passade även på att uppfinna ett eget filsystem i samband med detta. Ungefär vid denna punkt började han på allvar överväga att utveckla detta embryo till ett fullständigt operativsystem.

[8] En Intel 80386-baserad DX, 33MHz med 4 megabyte RAM-minne utan matematikprocessor (Intel 80387).

Linus kallade sitt operativsystem för *Linux*, men övervägde till en början att byta namnet till *Freax*. Vännen Ari Lemmke skapade dock en FTP-katalog åt projektet på datorn ftp.funet.fi som senare skulle användas för projektet, och döpte denna till */pub/OS/Linux*, och därmed etsades detta namn fast. Söndagen den 25 augusti annonserade Linus så slutligen projeketet:[9]

> Hej allihopa där ute som använder Minix —
>
> Jag bygger ett (fritt) operativsystem (bara som hobby, det kommmer inte att bli stort och professionellt som GNU) för 386 (486) AT-kloner. Detta har legat och pyrt sedan i april, och börjar bli färdigt. Jag skulle uppskatta all återkoppling rörande saker som folk gillar/ogillar i Minix, eftersom mitt OS påminner om det (samma fysiska struktur på filsystemet (av praktiska skäl) bland annat).
>
> Hittills har jag portat BASH (1.08) och GCC (1.40), och saker och ting tycks fungera. Detta innebär att jag kommer att få till något användbart inom något fåtal månader, och jag skulle vilja veta vilken funktionalitet folk skulle vilja ha. Alla förslag är välkomna, men jag lovar inte att implementera dem :-)[34]

I september publicerade Linus den första officiella versionen av Linux, version 0.01. Sedan följde en rask rad versioner och ett explosivt intresse för Linux. Version 1.0 släpptes dock inte förrän i mars 1994, men då hade ”terminalemulatorn” Linux redan fungerande TCP/IP-nätverksstöd, Ext2-filsystemet, matematikprocessoremulering, stöd för IDE- och SCSI-hårddiskar, stöd för ELF-binärer, uppringd förbindelse med SLIP och PPP, Ethernet via ett fåtal nätverkskort, NFS, avläsning av CD-ROM-skivor, datormusar, och ljud via ljudkortet Sound Blaster.

Version 1.0 täcktes också av GNU General Public License[10], som Torvalds valt eftersom den användes av hans favoritkompilator, GCC.

Linuxkärnan var redan från början helt oförmögen att hantera grafik: denna funktionalitet har hela tiden varit överlåten på fönstersystemet *X*, och därför inskränker sig kärnans stöd för grafikkort enbart till sådant som går utöver det som behövs för ren grafik, exempelvis accelererande avkodningskretsar för video eller ljud, eller OpenGL-gränssnitt.

I version 2.0 av Linux som släpptes i juni 1996 hade kärnan mognat betydligt, och hade stöd för dynamiskt laddningsbara moduler (se avsnittet om arkitektur längre ned), Pentiumprocessorn, nätverk med

[9] Min översättning.

[10] GPL, se sidan 122.

IPX, Appletalk eller paketradio (AX.25), betydligt fler nätverkskort och förutom Ethernet även Token Ring-nätverk, ISDN, flera nya filsystemtyper som SMB (Windows nätverksfilsystem) och VFAT (som används i Windows 95 och 98), samt många olika ljudkort.

Sedan dess har kärnan vuxit bit för bit via de stabila versionerna 2.2, 2.4 och 2.6. Det som tillkommit sedan 2.0 är främst stöd för ”hotplugging” (d.v.s. att hårdvara kan anslutas och tas bort under drift) speciellt för USB (Universal Serial Bus) samt *mer av allt* så att 2.6-kärnan stödjer i stor sett all förekommande hårdvara. Marknaden för hårdvaror har samtidigt mognat så att ett fåtal leverantörer (i realiteten ett fåtal integrerade kretsar och ett fåtal protokoll) har blivit dominerande, och används mer eller mindre i alla IBM PC-kompatibla produkter. Därför är det numera ganska lätt att få tag i hårdvara som fungerar tillsammans med Linuxkärnan.

Då detta skrivs är 2.6-versionen av kärnan ännu relativt ny, men jag har försökt att i görligaste mån beskriva handhavandet så att det skrivna gäller för båda kärnorna. I de fall detta inte är möjligt för att kärnorna skiljer sig åt, har jag försökt hålla mig till att skriva ut båda alternativen.

5.3 BSD

University of California, Berkeley hade i form av Berkeley Standard Distribution, kort kallad *BSD*, vidareutvecklat AT&T:s UNIX med utgångspunkt från källkoden till den sjunde versionen av UNIX från 1979. I början av 1990-talet släppte de källkoden till BSD-distributionen fri, och skapade därmed de fristående BSD-distributionerna baserad på det ursprungliga BSD:s kärna, kompilator och C-programbibliotek.

BSD portades snart till Intels 80386-processor i form av *386BSD*, ett projekt som annonserades av Bill Jolitz i *Dr. Dobbs Journal* i januari 1991. Under denna tid föregick en rättstvist mellan AT&T och University of California, Berkeley om rättigheterna till UNIX-källkoden, vilken dock till sist avgjordes till Berkeleys fördel.

Till en början var BSD:s licens inkompatibel med GNU GPL, men som beskrivs i avsnitt 3.4.2 på sidan 124 ändrades detta i juli 1999. Redan innan dess hade många användare av GNU/Linux börjat använda delar av BSD-programmen, i synnerhet distributionerna som ville vara så kompletta som möjligt. I andra fall har utvecklarna nöjt sig med att till exempel bara kopiera manualsidor, eftersom det varit jobbigt att skriva om dokumentation som redan fanns skriven för BSD.

Mängden program i ett normalt GNU/Linux-system som kommer från BSD-världen lär knappast öka, men däremot ökar mängden fri-

stående program som är tillverkade så att de fungerar både under GNU/ Linux och olika BSD-derivat. fönstersystemet *X* och de olika skrivbordshanterare som utvecklats, t.ex. GNOME och KDE, är utformade så att de fungerar under både BSD och GNU/Linux.

BSD-användarna använder på samma vis stora mängder program från GNU/Linux, men BSD-systemens kärna, C-bibliotek och större delen av kommandoradsverktygen är helt egna, modifierade versioner från sjunde versionens UNIX. Ett annat vanligt program från BSD-världen är OpenSSH, som har sitt ursprung i OpenBSD men som finns installerat i nästan alla GNU/Linux-system.

5.4 Arkitekturen i GNU/Linux-systemen

Linuxkärnan har i princip fem stora huvudkomponenter, samt en del mindre. De fem viktigaste är:[11]

Minneshanteraren eller **mm**[12] som tar hand om tilldelning och frigöring av minne för alla processer, samt hanterar virtuellt minne så att minnesbitar som inte används kan kopieras ut till hårddisk, d.v.s. det som i Linuxvärlden kallas *swap*, och som du normalt har skapat en speciell partition för under installationen.

Schemaläggaren eller **sched** som byter tidsluckor mellan olika processer på det vis som beskrivs i avsnittet om processer på sidan 33, och även hanterar system med mer än en mikroprocessor.

Interprocesskommunikatorn eller **ipc** som sköter kommunikation mellan körande processer.

Det virtuella filsystemet — kallat **VFS** eller **fs** — som har hand om alla filsystem och deras drivrutiner. Eftersom "allt är en fil" i POSIX-världen är detta den kanske största och mest komplexa biten av kärnan.

Nätverket eller **net** som hanterar drivrutiner för nätverkskort, TCP/IP och paketfiltrering.

Linuxkärnan är, som tidigare sagts inte en mikrokärna, utan en monolitisk kärna. Skälen till detta anges av Linus Torvalds vara[24]:

[11] Kapitlet presenterar en övergripande och förenklad bild av Linuxkärnan så som den upplevs av en användare, för en djupgående och detaljerad presentation avsedd för programmerare, se standardverket *Linux Kernel Development*[20].

[12] De olika delarna av kärnan har kortnamn som härleder sig från de kataloger i källkodsträdet där motsvarande källkod ligger.

- Mikrokärnor var, när han började skriva Linux, ännu experimentella.
- De var av naturen mycket komplexare.
- De var också långsammare.

Redan år 1992 försiggick en hård men hjärtlig debatt i Usenet-gruppen comp.os.minix mellan Linus och Andrew Tanenbaum om för- och nackdelar med mikrokärnor (som i Minix) versus monolitiska kärnor (som Linux). Den som är intresserad kan ju läsa detta,[13] men historien har bevisligen redan givit Torvalds rätt i denna debatt. Linus skriver med anledning av bland annat detta:[14]

> Det var faktiskt så, att detta fick mig att anse att mikrokärnemetodiken i grunden var en ohederlig metodik, inriktad mot att erhålla fler dollar för forskning. Jag tror inte att dessa forskare nödvändigtvis var medvetet ohederliga. Kanske var de bara dumma. Eller förda bakom ljuset. Jag menar verkligen detta.[24]

Personligen anser jag att kärnan skall uppfattas som en hybrid mellan mikrokärna och monolitisk kärna. Anledningen är att vissa delar av kärnan, såsom loggmodulen, faktiskt startas som självständiga processer utanför kärnan. (Se vidare avsnitt 5.4.4 på sidan 194.)

Att kärnan inte är någon mikrokärna har inte hindrat Linux från att vara *portabelt* och *modulariserat*.

Portabiliteten i Linux innebär att systemet kan flyttas mellan olika datorarkitekturer utan omfattande ändringar, och initierades egentligen genom en port till Motorola 68000-familjen (bland annat Amiga), men blev inte ordentligt genomförd förrän David Miller inledde ett experiment med att porta Linux till Sun SPARC-arkitekturen. Större delen av Linux är så abstrakt som möjligt och skrivet helt i programspråket C, med endast mindre bitar i assembler. Funktioner för att hantera externa enheter (drivrutiner) är i allmänhet skrivna så att det inte spelar någon roll vilken processor eller buss som används i datorn där kärnan installeras: drivrutinen kan användas i alla fall. Exempelvis borde ett godtyckligt PCI-kort för IBM PC-kompatibler — *i teorin* — kunna användas i en Linuxkärna installerad på en Macintosh med PCI-buss. Assemblerkodgenerering och optimering överlåts istället mestadels på GNU C Library och GNU C Compiler. Detta har lett till att Linux stödjer den stora mängd arkitekturer som återges i figur 5.3

[13] Se [35].

[14] Min översättning.

DEC Alpha	26-bitars ARM
H8/300-familjen	IA64 (Itanium)
Motorola 68000-varianter utan MMU	PA-RISC
64-bitars PowerPC	SuperH
64-bitars Sun SPARC	NEC V850E
32-bitars ARM	Axis CRIS/ETRAX
Intel 386-varianter†	Motorola 68000-varianter
MIPS	PowerPC
IBM S/390	Sun SPARC
User Mode Linux (Linux i virtuell maskin)	x86-64 (Opteron)

Figur 5.3: Datorarkitekturer som stöds av Linuxkärnan version 2.6. † = inkluderar Intel 80386, 80486, Pentium, Pentium-II och Pentium-III $\pm$ MMX, 3DNOW, även kompatibla processorer som Transmeta Crusoe, AMD o.s.v. Notera den svenska arkitekturen CRIS som används i ETRAX-processorn från Axis Communications.

Modulariteten i Linux introducerades med Linux 2.0 och innebär att *moduler* med funktionalitet, exempelvis drivrutiner för hårdvara, kan väljas vid kompileringen av kärnan. Du kan alltså nöja dig med att bygga moduler för enbart den hårdvara som finns i din dator. Det kan vara värt att notera att svensken Björn Ekwall, som arbetat med Linuxkärnan sedan innan den var i version 1.0, har programmerat stora delar av modulhanteringssystemet.

Linux har också stöd för dynamiskt inladdade moduler: detta innebär att moduler kan laddas in och ut ur kärnan under drift *alternativt med* att kompileras in så att de alltid finns i kärnan. När du kompilerar din kärna, kan du välja att kompilera stora mängder av moduler som inladdningsbara om du har en känsla av att de *kanske* kommer att behövas, och de är därmed tillgängliga att användas vid behov. (Exempel: jag behöver *kanske* SCSI, NFS o.s.v.) Du kan också välja att helt stänga av funktionen för inladdningsbara moduler, vilket du typiskt skall göra på ett inbyggt system som ju har en fix och färdig hårdvara som inte ändras med tiden.

Verktyg för att styra och kontrollera kärnan följer inte med installationen av själva kärnan som sådan; istället krävs externa paket med verktyg som kommunicerar via de standardiserade gränssnitten för att hantera kärnan.

I distributionerna har distributören i allmänhet valt att kompilera in de mest grundläggande modulerna i kärnan, medan de mer perifiera eller obskyra har kompilerats som inladdningsbara moduler. Oftast finns i princip alla upptänkliga moduler med vid leveransen av en distribu-

tion, det enda undantaget är moduler som skulle vålla konflikt med en annan modul. I sådana fall brukar den *vanligaste* av de moduler som hamnar i konflikt levereras med distributionen.[15]

Kärnan är också relativt ren från dåliga arv: dumheter som att bara 11 tecken kan användas i ett filnamn, eller att installation av drivrutiner för hårdvara kräver att datorn startas om, lyser med sin frånvaro i Linux. Detta beror till lika delar på att Torvalds lyssnat på många goda råd innan han spikat en design, och att implementationen påbörjades så pass sent.

Torvalds har också en tendens att anamma en attityd som innebär att det som är dåligt gjort från början inte skall "fixas till" utan göras om helt från grunden, så att delarna passar ihop på bästa vis.[16] Detta står i bjärt kontrast med den deadline-orienterade metodik som används på många mjukvaruföretag — en sådan metodik genererar ofta ett bålverk av underligheter som både utvecklare och användare sedan måste tas hänsyn till under lång tid, något som tyvärr ofta också stoppar upp vidareutvecklingen. Här är Linuxkärnan ett föredöme med sin rena och lättbegripliga arkitektur.

5.4.1 Hantering av dynamiska kärnmoduler

Själva modulerna till Linuxkärnan ligger i allmänhet i katalogen */lib/modules/x.y.z* där *x.y.z* är versionen av kärnan som modulerna hör till.[17] Du har i allmänhet bara en enda kärna installerad på din dator, men det är inget som hindrar att du har flera. Oftast är det då olika versioner av kärnan, eller olika versioner av samma kärna som det är fråga om. Varje ny kärna du installerar skall dock ha ett unikt namn.

I denna katalog ligger modulerna ordnade i olika kataloger som svarar mot vilken *typ* av modul det är fråga om, t.ex. *net* (nätverk), *fs* (filsystem), *ipv4* (IP-protokollet version 4), och i de här katalogerna finns i sin tur modulerna med namn som *nfs.o* (NFS), *sb.o* (Sound Blaster) o.s.v. Att filerna har suffixet *.o* betyder inget annat än att de är objektfiler, d.v.s. kompilerad kod, och själva *modulnamnet* är således *nfs*, *sb* etc. I version 2.6 av kärnan heter filerna istället samma sak med ändelsen *.ko* (kernel object), t.ex. *nfs.ko*.

Följande verktyg följer med i varje GNU/Linux-distribution eller kan kompileras från källkodspaketet *Modutils* (för 2.4.x-serien) eller *Module-init-tools* (för 2.6.x-serien), och hanterar dynamiskt inladdade moduler:

[15] De flesta av dessa konfliktproblem har lösts i version 2.6 av Linuxkärnan.

[16] Det svenska talesättet *gör om, gör rätt* passar ganska bra in på denna metodik, i programvarubranschen kallas angreppssättet för *refaktorering*.

[17] För exakt versionsangivelse för mer exotiska varianter, se avsnitt 3.5 på sidan 126.

modprobe foo lägger till modulen *foo* i kärnan under drift. Modprobe är också det kommando som i allmänhet *skall* användas för att såväl lägga till som ta bort moduler i kärnan: det är ett högnivåkommando, som i sin tur använder kommandona **insmod** och **depmod** (se nedan).

Vad modprobe tillför är att det räknar ut om en modul är beroende av någon annan modul (på samma vis som olika paket i en distribution kan vara beroende av varandra t.ex.) och sedan laddas de i ordning.

En modul som du lagt in med **modprobe foo** kan tas bort igen med **modprobe -r foo**.

Modprobe kan också ta hand om speciella saker som skall göras före och efter installationen av en modul, eller om några speciella parametrar skall skickas med till modulen.

I filen */etc/modules.conf* (för 2.4.x-serien) eller */etc/modprobe.conf* (för 2.6.x-serien) finns en lista med konfigurationer för olika moduler. Här listas parametrar (engelska: option) och liknande som hör ihop med modulerna. Vissa distributioner har automatiska verktyg för att underhålla den här filen och då skall du normalt inte ändra i den för hand. I annat fall är det helt fritt fram.

Om du använder en distribution finns det naturligtvis något skript som har hand om att se till att de moduler du valt under installationen laddas in under uppstart och finns inladdade när systemet startat. I Debian GNU/Linux finns en lista i */etc/modules* över de moduler som skall läsas in, och i Red Hat eller Fedora Core görs det successivt i skriptet */etc/rc.d/rc.sysinit*. Någon standard för hur detta skall gå till existerar inte, men du kan antagligen hitta rätt i din distribution genom att utgå från raden som börjar *si::sysinit:...* i */etc/inittab*.

lsmod ger en lista över vilka moduler som är installerade i kärnan just nu, om de används, och i så fall hur många processer som använder just den modulen. Prova gärna att köra detta för att se vilka moduler din hårdvara egentligen använder.

modinfo foo ger information om hur modulen *foo* används: vilken fil den kom ifrån och och vilka parametrar den startats med.

insmod är ett mera brutalt kommando som kan användas istället för **modprobe** för att helt enkelt trycka in en modul i kärnan. Detta kräver ett fullständigt filnamn, t.ex. **insmod /lib/modules/2.4.20/**

drivers/foo.o Kommandot insmod används normalt bara av programmerare, en "vanlig" användare bör använda modprobe.[18]

rmmod foo plockar bort modulen *foo* ur kärnan. Du kan inte plocka bort en modul som används. Även detta kommando används i sin tur av modprobe.

Tänk på att du absolut måste vara root för att köra dessa program, och att de kanske inte är tillgängliga som standard i ditt skal: Modutils ligger normalt installerat i */sbin* i ditt filsystem, så kanske måste du skriva hela sökvägen, exempelvis: **/sbin/lsmod** för att använda dem.

5.4.2 Hur kärnan startar

Vissa delar rörande hur datorn startar upp har redan beskrivits under avsnittet om bootstrap loader, sidan 138. Din bootstrap loader kommer att ge dig någon form av *bootprompt*, antingen i form av en meny med förvalda alternativ, eller eventuellt i form av en ren kommandorad, som ser ut något i stil med:

```
LILO:
```

Med en sådan här prompt kan du ange vilken kärna du vill starta, kanske skriver du bara **linux**, eventuellt ger du några parametrar till kärnan innan du startar den, t.ex. **linux single** för att starta kärnan i *enanvändarläge* vilket är detsamma som körnivå 1 (se avsnitt 5.4.3). Vanligast är dock att du bara trycker på Enter-tangenten, vilket aktiverar ett fördefinierat alternativ. I GNU GRUB används uteslutande menyrader för de olika konfigurationerna, och då kan du istället välja att redigera en av konfigurationsraderna, nämligen den som börjar med något i stil med *kernel (hd0,0)/boot/...* och på det viset ge parametrar till kärnan [19]

För att starta Linuxkärnan behöver din bootstrap loader oftast två olika filer, ibland bara en:

- En **kärnavbild** som innehåller själva Linuxkärnan. Den har ofta namn som *vmlinux* eller *vmlinuz*, eller ett längre, fullständigt namn. Den kan också vara en länk till den egentliga filen, så länge denna ligger i samma filsystem.

[18] **insmod** är egentligen kommandot som gömmer sig bakom både **modprobe**, **lsmod** och **rmmod**, eftersom alla dessa är symboliska länkar till insmod. Programmet insmod känner sedan i sin tur av hur det har anropats, och beter sig därefter.

[19] I själva verket ges bara en del av parametrarna kärnan, och resten till den s.k. *init*-processen, som är den första process som startas i GNU/Linux.

Namnet *vmlinux* kommer från UNIX-varianter där motsvarande fil har namnet *vmunix*. *vm* står förmodligen för *virtual machine* och i det ena fallet är *x* utbytt mot *z* för att markera att filen är zippad med **gzip**.

Denna fil kallas också *zImage* eftersom det normalt är en gnuzippad kärna. Tidigare kunde LILO bara ladda 1 megabyte stora kärnor, men i och med att denna begränsning försvann i bootstrap loader-programmen började denna fil istället att kallas *bzImage* från engelskans *big zipped image*. (Ett vanligt missförstånd är att detta skulle stå för att kärnan är komprimerad med det modernare komprimeringsprogrammet bzip2, så är det alltså *inte*.)

Hela kärnavbilden är inte komprimerad: den innehåller också ett par program skrivna i assembler för att få igång den mest grundläggande hårdvaran och genomföra själva dekomprimeringen av resterande kärna.

- En initial *RAM-disk* som innehåller ett minimalt filsystem som skall användas för att starta systemet. Detta filsystem skapas inte på någon disk utan direkt i RAM-minnet på maskinen och kallas därför just RAM-disk. Efter att det använts kommer det att kastas bort igen.

  Denna RAM-disk innehåller typiskt bara några moduler som skall laddas in i kärnan. Om du väljer att kompilera in alla moduler du använder direkt i kärnan så behövs ingen initial RAM-disk alls.

  Om din distribution har installerat en initial ramdisk kan du alltid undersöka den lite (som root): kopiera den med t.ex. **cp /boot/initrd.img ./copy.img.gz** packa upp den med **gunzip copy.img.gz** och undersök innehållet genom att montera avbilden med något i stil med **mkdir initimage ; mount -t ext2 -o loop copy.img initimage**[20]

  På RAM-disken finns en fil som heter */linuxrc* och som körs automatiskt av kärnan under uppstarten.

Dessa filer pekas ut i konfigurationsfilerna till LILO eller GNU GRUB, genom att ange en partition där de ligger,[21] och därefter *vart* på denna partition de två filerna finns. Till exempel kan du skriva i *grub.conf*:

[20]För information om s.k. *loopback-montering* av en diskettavbild, se sidan 353.

[21]Det vill till att din bootstrap loader kan hantera det filsystem som används på denna partition (eftersom kärnan ännu inte är startad när den laddas in), därför används vanligen Ext2-filsystemet. Detta är också den huvudsakliga anledningen till att ett separat */boot*-filsystem ofta används till detta — */boot* behöver inte användas annat än av bootstrap loadern.

```
root (hd0,0)
kernel (hd0,0)/vmlinuz rw root=/dev/hda1
initrd (hd0,0)/initrd.img
```

Detta anger att den primära IDE-hårddiskens första partition (hd0,0) innehåller ett filsystem. Detta skall monteras, och kärnan startas från filen *vmlinuz* med RAM-disken *initrd.img*.

Parametrarna till kärnan har formen *foo=bar fnord=p1,p2,p3* o.s.v., var noga med att inte infoga några mellanslag i en parameterlista separerad med komman. I exemplet har parametern *foo* värdet *bar* medan *fnord* har värdet *p1,p2,p3*. Det finns några användbara parametrar du kan använda:

root=/dev/hda1 talar om att Linuxkärnan skall använda partitionen */dev/hda1* som rotpartition, d.v.s. för att montera katalogen med namnet /.

Detta är *inte* samma sak som den root som anges för bootstrap loadern, d.v.s. i exemplet ovan hd0,0. Det senare är avsett att vara en ”boot-root” som skall monteras för att läsa in kärnan och RAM-disken, varken mer eller mindre.

single begär att kärnan skall starta i enanvändarläge med en root-prompt. Detta är mycket bra om datorn till exempel har kraschat och behöver repareras.

ro monterar rotfilsystemet (/) i *read-only*-läge, vilket innebär att inget på disken kan ändras. Om du har en dator som bara används som brandvägg kan detta på ett finurligt vis hindra den som lyckas ta sig in i systemet från att sedan ändra något i systemet. Nackdel: se till att dina systemloggar (d.v.s. normalt filhierarkin under */var*), inte ligger i rotfilsystemet om du gör detta, annars blir det inga loggar. */tmp* får helst inte heller ligga där, eftersom många program behöver mellanlagra saker i */tmp*.

Många distributioner, t.ex. Red Hat, väljer att först montera rotfilsystemet med denna flagga, och sedan *montera om* rotfilsystemet läs/skrivbart, när uppstartskripten kontrollerat att filsystemet verkligen mår bra. Detta är bra, eftersom det gör det möjligt att åtgärda fel i rotfilsystemet, något som uppstår om du t.ex. stänger av strömmen till datorn utan att stänga av den ordentligt. Sådana fel kan bara åtgärdas på ett filsystem som monterats i *read only*-läge.

rw tvingar istället kärnan att montera rotfilsystemet både läs- och skrivbart.[22]

rootflags=foo kan ta alla parametrar som du kan ge till kommandot **mount -o ...**, se sidan 213. Dessa kommer att användas när rotfilsystemet monteras.

console=ttyS1,9600 kommer att flytta systemkonsollen till serieport nr 2 och försöka kommunicera där med hastigheten 9600 kbit/sekund. Om du inte har en skärm till din dator kan det finnas tillfällen då du behöver ansluta en terminal istället, och detta är sättet det görs på.

reboot=w säger till kärnan att göra en varmstart (d.v.s. inte räkna upp minnet och liknande) när en omstart begärs av en användare. I standardfallet görs en kallstart, beroende på att gammal hårdvara inte kunde hantera varmstarter ordentligt.

vga=ask ber kärnan att presentera en lista av tillgängliga videolägen när den startar. Flaggan kan även sättas "hårt" till ett videoläge som du gillar, t.ex. **vga=7**, men du vill nog veta vad den siffran betyder. Detta får du veta genom att köra **ask** en gång först.

Dessa parametrar är bara några av det femtiotal som stöds av kärnan, men de övriga är alla relativt obskyra och avsedda för folk som utvecklar Linuxkärnan.

När den komprimerade Linuxkärnan och den eventuella RAM-disken laddats in från utpekad plats kommer kärnan att "patchas" genom att vissa parametrar skrivs in i fördefinierade minnesadresser. Sedan startas Linuxkärnan. Detta tillstånd brukar ibland kallas körnivå noll (engelska: runlevel zero), men den beteckningen är något tvivelaktig. Det är nu kärnan börjar spruta ur sig den formliga flod av obegriplig text som kännetecknar många GNU/Linux-installationer.

Om vi börjar från det ögonblick som LILO eller GNU GRUB lämnar över kontrollen till kärnan händer följande på en IBM PC-kompatibel dator:[23]

- Kärnan startar en assemblerkodsnutt som passande nog har namnet *trampoline* och initierar viss hårdvara. Bland annat väljs där en

[22]Du kanske undrar vilken av flaggorna **ro** och **rw** som är standardalternativet. Det beror på. Detta tattueras in i kärnavbilden efter att den kompileras och kan vara både det ena och det andra. Men **rw** är nog det normala.

[23]Förr kunde Linux starta sig självt via ett eget bootblock. Denna funktionalitet är numera helt överlåten till externa bootstrap loaders. Här beskrivs hur en *bzImage* laddas in och startas, för mer detaljer se [28].

videoläge för startup, vilket är det som kan ställas in med **vga**-parametern till kärnan.

- Den komprimerade delen av kärnan har ett inledande dekomprimeringsprogram som sedan startas i skyddat läge (engelska: protected mode) vilket är det processorläge som en kärna normalt skall köras i. Detta program packar upp den komprimerade delen av kärnan.
- Efter detta hoppar kärnan till den dekomprimerade kärnavbilden. Här startas en minimal assemblersnutt som anropar C-funktionen start_kernel().
- Kärnan kommer därefter att initiera sina huvudsakliga komponenter (schemaläggaren, minneshanteraren, det virtuella filsystemet och interprocesskommunikationsdelen), samt alla de moduler som kompilerats in i kärnan. Detta genererar huvuddelen av den textmassa som far över skärmen vid uppstart. Efter att uppstarten är klar lagras denna text i */var/log/dmesg* men du kan också se uppstarttexten med kommandot **dmesg** som även visar meddelanden från kärnan som kommit till efter själva uppstarten.
- Om du har skapat en RAM-disk till din kärna kommer den sedan att monteras och skriptet **linuxrc** i roten av det filsystemet att köras. Till sist monteras rotfilsystemet /.

Efter att kärnan har startat kommer den automatiskt att försöka köra programmet **/sbin/init**[24] från rotfilsystemet för att starta den allsmäktiga ursprungliga *init*-processen. Byter du ut programmet som ligger där mot något annat, så kommer detta att startas istället. (Gör *inte* detta experiment!) *init* kommer att startas i *usermode*, vilket innebär att den inte kan kommunicera med kärnan annat än genom systemanrop, och om *init* skulle krascha innebär detta alltså att själva kärnan inte behöver krascha av det (även om det säkert skulle upplevas på det viset).

Efter att init startats har denna process fullständig kontroll över datorn och operativsystemet, kärnan förhåller sig passiv, och tar inga egna initiativ. Det enda den gör är att ta emot kommandon från *init* och andra processer, samt olika former av servicebegäran från datorns hårdvara, såsom avbrott från olika former av instickskort, stoppursbaserade avbrott, meddelanden om avslutade DMA-aktiviteter,[25] o.s.v.

[24]Om inte **/sbin/init** finns kommer kärnan att försöka med **/etc/init** och till sist med **/bin/init**. Sedan ger den upp och försöker starta **/bin/sh** för att erbjuda ett skal där felet kan åtgärdas. Du kan också ange ett speciellt *init*-program med parametern **init=/bin/foo** till kärnan i din bootstrap loader.

[25]DMA, *Direct Memory Access* är en metod för hårvaran att kommunicera direkt med datorns minne utan att blanda in processorn.

Inladdningen av moduler som inte kompilerats in i kärnan sköts till exempel av *init* (eller andra processer) men inte av kärnan själv. På sätt och vis blir kärnan till ett enda stort anropbart programbibliotek.

Det vanligaste *init*-programmet kallas *SysVinit* (utläses *system five init*) av Miquel van Smoorenburg. Detta har stora likheter med andra POSIX-system och använder de s.k. *körnivåerna* för att styra systemets tillstånd. En del tycker att detta är ett avskyvärt komplext program, och kör istället en ”light-variant” med namnet *simpleinit*. Du kan för all del skriva ett eget *init*-program om du hellre vill det.

De som arbetar med inbyggda system kan köra vilket program som helst istället för det klassiska *init*. Många inbyggda system startar en multitrådad applikation med ett grafiskt gränssnitt, som inte använder grafikrutinerna från fönstersystemet *X*.

5.4.3 Kärnkontroll — körnivåer

För att kontrollera operativsystemets tillstånd i GNU/Linux används för det mesta så kallade *körnivåer* (engelska: *runlevels*). (Om ditt system inte använder dem är följande information inte särskilt intressant.) *SysVinit* är ett program som implementerar denna funktionalitet.

Dessa körnivåer anger i lösa ordalag ”hur mycket av operativsystemet som startats”. Alla GNU/Linux-distributioner använder *inte* dessa körnivåer, och vissa andra POSIX-system som exempelvis Solaris *använder* dem. Det är ett kärt ämne för groll och dispyt bland GNU/Linux-användare hurvida systemet med körnivåer är bra eller dåligt. Vissa avskyr dem, andra älskar dem.

Utöver dessa körnivåer finns oftast ett *bootskript* som bara körs *en gång* när datorn startas. (Jämförbart med det skript kan ligga på ramdisken). Vart detta skript finns varierar mycket mellan distributioner och installationer men det vanligaste är */etc/rc.sysinit* eller */etc/init.d/rcS*.[26] Bootskriptet brukar t.ex. ta hand om att avmontera ramdisken, sätta parametrar till kärnan från */etc/sysctl.conf* och ladda in nödvändiga moduler i kärnan innan något annat utförs.

När du startar ditt GNU/Linux-system bootas kärnan, och när sedan *init*-processen[27] startas syns följande text på skärmen:

```
INIT 2.85 booting:
```

Körnivåerna utgår från att den första process som startas utanför

[26]Du kan ta reda på vad det heter i just ditt system genom att i filen */etc/inittab* leta reda på raden som ser ut något i stil med: **si::sysinit:/etc/rc.d/rc.sysinit**

[27]D.v.s. programmet SysVinit.

kärnan består av processen *init*. Denna process har ingen förälder[28] och startar i sin tur alla andra program gruppvis, beroende på vilken körnivå som begärts. Alla processer på ett GNU/Linux-system är således släktingar till *init* på ett eller annat vis. (*Init* har själv processen med process-ID *0* som förälder, en process som inte existerar.)

Om du vill se hur detta släktskap ser ut på just ditt system kan du skriva kommandot **pstree**, som ger en grafisk bild av hur föräldrar och barn är släkt med varandra, med *init* som ursprungsprocess.

Init har hand om logistiken för alla processer i systemet, ser till att de lever när de skall leva, och begraver dem när de dör.

Livscykeln för processer i GNU/Linux är följande: om processer dör skall de tas bort av sina föräldrar. Detta går till så att när en process har dött (avslutats) av någon anledning, förvandlas den först till en zombieprocess. (En zombie är en ”levande död” inom voodoo-religionen.) Därefter skall föräldern rensa bort (”begrava”) processen. Om detta inte fungerar som det skall, och förälderprocessen dör innan den hunnit rensa bort sina zombiebarn, kommer *init* att överta ägarskapet till den föräldralösa processen och om det är en zombieprocess kommer *init* sedan att rensa bort den. Detta går alltså till på samma sätt som när staten begraver de människor som dör men som saknar levande släktingar. (Lyckligtvis slipper de flesta föräldrar i verkligheten att begrava sina barn.) Om en föräldralös process *inte* är en zombieprocess, kommer den istället att leva vidare i högönsklig välmåga och är därmed istället en *demon*.

Alla processer som inte startas av en användare, skall startas av *init*. Varje sådan process skall vara definierad i en *körnivå*.

Körnivåerna var från början tänkta att vara nio till antalet, men bara sex finns definierade. Betydelsen av dessa är inte exakt fastslagen och kan variera mellan olika system, men för det mesta har de en betydelse som den i figur 5.4.

Vilka processer (främst demoner) som startas i vilken körnivå är ingen större hemlighet utan anges i filen */etc/inittab*, där du med lite klåfingrighet kan omdefiniera helt vad körnivå-siffrorna skall betyda.[29] Här finns även angivet vilken körnivå som systemet kommer att gå upp i som standard (normalt nivå 5). Du kan vilja ändra denna till 4 för att datorn enbart skall starta i textläge, exempelvis för en server av något slag. En annan sak som kan kontrolleras i denna fil är hur systemet skall reagera på tangenkombinationen Ctrl+Alt+Delete, känd från bland annat Windows-inloggningen.

---

[28] Repetera POSIX-kapitlets avsnitt om processer på sidan 33 om du glömt terminologin för processer!

[29] Red Hat har gjort ett program som heter **chkconfig** som kan hjälpa till att avlusa *inittab*-filen. Om det finns installerat med din distribution ligger det i **/sbin/chkconfig**.

0 stoppa systemet, stopp, halt. Synonym för avstängt operativsystem. Att växla till denna körnivå stänger ned hela operativsystemet. Används också som en beteckning på tillståndet i kärnan innan *init*-processen startats under uppstart.

1 enanvändarläge, synonym **S** eller **s** (båda fungerar som kommandoargument). Systemet monterar här om rotpartitionen i läs/skrivbart läge (från att enbart ha varit läsbart), monterar sedan alla andra lokala filsystem, och aktiverar swap-partitionen för virtuellt minne. Moduler till kärnan laddas och tangentbordet konfigureras. En ensam terminal (konsollen) startas också. I detta läge har systemet endast en användare, på värdmaskinens konsoll. Loggningsdemonerna **syslogd** och **klogd** startas normalt här.

2 fleranvändarläge *utan* nätverk. Ett antal terminaler startas som kan användas oberoende av varandra.

3 fleranvändarläge *med* nätverk. Datorn får ett IP-nummer på något vis och ansluter till nätverksenheter.

4 fleranvändarläge med textanvändargränssnitt aktiverat. I Linux Standard Base definierat som ”reserverat för lokala anpassningar”.

5 grafiskt fleranvändarläge, här startas oftast bara X-servern och en skärmhanterare, t.ex. XDM (se kapitel 6), förutom de processer som startats i lägre nivåer.

6 omstart — att växla till körnivå 6 kommer att starta om hela systemet.

Figur 5.4: Olika körnivåer.

```
#!/bin/sh
level=$1
cd /etc/rc.d/rc$level.d
for i in K*; do
    $i stop
done
for i in S*; do
    $i start
done
```

Figur 5.5: Det skript som växlar mellan olika körnivåer. Skriptet går in i rcX.d-katalogen för aktuell körnivå, och kör först alla skript som börjar på **K** i alfabetisk ordning och sedan alla skript som börjar på **S** i alfabetisk ordning.

Programmen kan oftast inte startas rakt av utan använder i sin tur olika skriptfiler, vilka brukar placeras i katalogen */etc/init.d*.[30] För att *init* skall veta vilken körnivå respektive skript hör till placeras symboliska länkar från kataloger i */etc/rcN.d*[31] där **N** är respektive körnivå. (rc0.d, rc1.d o.s.v.) Dessa kallas föga förvånande för *initskript*.

Länkarna som placeras i */etc/rcN.d* skall ha ett prefix som inleds med **S** (stora S) eller **K** (stora K) samt följs av två siffror. Sifforna skall alltid vara två, så möjliga kombinationer är S00-S99 och K00-K99.

S anger att programmet (typiskt en demon) skall *startas* med detta könummer och **K** anger att processen skall *stoppas* (K kommer av engelskans *kill*) med detta könummer. K-skripten kommer att köras först, och skall vara symboliska länkar till initskripten för *alla* program som *över huvud taget* kan vara startade i någon annan körnivå, men som inte skall vara startade i denna nivå. Det kan alltså bli fråga om en hel del länkar. Efter detta körs S-skripten, som startar de demoner som *skall* köra i denna nivå. Numret anger körordning och är praktiskt att ha till hands när du vill starta eller stänga ner demoner som är beroende av varandra. Om **foo** är beroende av att **bar** är startat så namnger du exempelvis länkarna **S10bar** och **S20foo** så startas det ena före det andra. Du behöver faktiskt inte ens välja en ledig siffra, bara namnet är unikt, och den andra demonen med denna siffra inte är beroende av din demon. Systemskriptet som går igenom *rcN.d*-katalogerna för varje körnivå ligger för det mesta i */etc/rc.d/rc* och om du tittat i detta skript och förstått det så inser du hur detta fungerar i praktiken, se figur 5.5.

Ett initskript för en demon **fnord** som bara skall köras i körnivå 4

[30] Utläses förmodligen *init directory*

[31] Utläses förmodligen *runlevel configuration directory*

prepareras således enklast genom att skriva ett bash-skript som tar parametrarna **start** eller **stop**, och kopiera detta till */etc/init.d/fnord*. Därefter länkar du detta program till exempelvis */etc/rc4.d/S50fnord*, så att det startas automatiskt i körnivå 4. *Men* glöm nu inte att göra kill-länkar till *alla* andra körnivåer, dvs länka */etc/init.d/fnord* till exempelvis */etc/rc0.d/K50fnord*, */etc/rc1.d/K50fnord*, */etc/rc2.d/K50fnord*, */etc/rc3.d/K50fnord*, */etc/rc5.d/K50fnord*, */etc/rc6.d/K50fnord*, om du nu vill att det verkligen bara skall vara startat i en enda körnivå.

Om du tittar i katalogen */etc/rc0.d* (som stoppar maskinen) så ser du att alla länkar i princip är kill-skript. Alla utom två: till **S00killall** och **S01halt**, som stannar maskinen helt efter att alla demoner stoppats.

Genom att titta i */etc/rcN.d*-katalogerna upptäcker du ganska snart hur det hela fungerar.

Medan *init*-processen kör kan du (eller rättare sagt root-användaren) skicka meddelanden till den om att byta körnivå. Detta görs med kommandot *init*, vilket inte skall förväxlas med själva *init*-demonen som sådan.

Följande kommandon används för att kontrollera *init*-demonen och kan normalt bara utföras av root-användaren (undersök deras växlar och egenheter med **man**):

init körnivå — där *körnivå* är en siffra eller bokstav från figur 5.4 — anger att demonen skall växla körnivå till den angivna. Exempelvis innebär kommandot **init 1** att maskinen omedelbart tas ned i enanvändarläge. Kommandot **telinit** finns också tillgängligt för att utföra samma sak, men är bara en länk till **init**. Anledningen är att andra POSIX-system använder detta istället för **init** direkt.

runlevel skriver ut vilken/vilka körnivåer som operativsystemet befann sig i senast, följt av vilken den befinner sig i nu. Om systemet inte har befunnit sig i någon annan körnivå än den aktuella sedan det startades, markeras föregående körnivå med ett *N*. Typiskt resultat av kommandot *runlevel*: ”N 5”. Detta kommando används knappt av fysiska användare, det används av systemet.

shutdown stänger ned systemet efter en viss tid, och efter att ha gett en skriftlig varning till samtliga inloggade användare. Utför i slutänden synonymen till kommandot **init 0**.

halt stannar systemet omedelbart. Synonym för kommandot **init 0**.

reboot startar om systemet omedelbart. Synonym för kommandot **init 6**.

Om parametern **respawn** har angetts för något program i */etc/inittab* innebär detta att initdemonen kommer att göra sitt yttersta för att hålla

```
USER       PID %CPU %MEM   VSZ  RSS TTY      STAT START   TIME COMMAND
root         1  8.9  0.0  1372  428 ?        S    19:44   0:04 init [S]
root         2  0.0  0.0     0    0 ?        SW   19:44   0:00 [keventd]
root         3  0.0  0.0     0    0 ?        SW   19:44   0:00 [kapmd]
root         4  0.0  0.0     0    0 ?        SWN  19:44   0:00 [ksoftirqd_CPU0]
root         9  0.0  0.0     0    0 ?        SW   19:44   0:00 [bdflush]
root         5  0.0  0.0     0    0 ?        SW   19:44   0:00 [kswapd]
root         6  0.0  0.0     0    0 ?        SW   19:44   0:00 [kscand/DMA]
root         7  0.0  0.0     0    0 ?        SW   19:44   0:00 [kscand/Normal]
root         8  0.0  0.0     0    0 ?        SW   19:44   0:00 [kscand/HighMem]
root        10  0.0  0.0     0    0 ?        SW   19:44   0:00 [kupdated]
root        11  0.0  0.0     0    0 ?        SW   19:45   0:00 [mdrecoveryd]
root        15  0.0  0.0     0    0 ?        SW   19:45   0:00 [kjournald]
root        69  0.0  0.0     0    0 ?        SW   19:45   0:00 [khubd]
root      2251  0.0  0.0     0    0 ?        SW   19:45   0:00 [kjournald]
root      2287  0.0  0.0  1364  320 ?        S    19:45   0:00 minilogd
root      2328  0.0  0.0  1372  428 tty1     S    19:45   0:00 init [S]
root      2329  0.0  0.2  2188 1156 tty1     S    19:45   0:00 /bin/sh
root      2331  0.0  0.1  2608  664 tty1     R    19:45   0:00 ps aux
```

Figur 5.6: Processer i körnivå 1 på Red Hat Linux 9. Ett ganska typiskt urval.

detta program vid liv. Om det stannar kommer det genast att startas om igen. Om ett program startar om för snabbt, mer än 10 ggr på 2 minuter, kommer den dock att ge upp och betrakta programmet som trasigt. (Detta drabbas du ibland av när init försöker starta en *X*-serverprocess, se avsnitt 6.3.)

För skripten i */etc/rcN.d*-katalogerna används inte ”respawn”, och du kan därför stoppa enstaka av dessa demoner med kommandot **kill** om du vill. Det normala sättet att stoppa en demon är dock genom att köra dess *init*-skript med parametern **stop**, t.ex.: **/etc/init.d/foo stop**. Du kan också starta den igen med **/etc/init.d/foo start**.

5.4.4 Exkursion i körnivå 1

För att skapa oss en uppfattning om vilka processer som krävs i ett GNU/Linux-system för att det skall ge en hyfsad funktionalitet, gör vi nu en utflykt till körnivå 1 i en normal distribution för att undersöka vilka processer som körs.

Du kan göra detta experiment själv genom att antingen ge växeln **single** till kärnan vid uppstart, eller växla till körnivå 1 med kommandot **init 1** som root. (Det senare kan dock ibland medföra att vissa extraprocesser lever kvar från högre nivåer.) Kör sedan kommandot **ps aux** för att se alla processer som kör i systemet för tillfället. **ps aux > foo.txt** sparar resultatet till en fil.

På min bärbara dator med Red Hat Linux 9 ger detta resultatet i figur 5.6.

Den första processen är den välbekanta *init* med process-ID 1, och [S]:et efter processen anger att *init* startats i enanvändarläge.

De tre sista processerna körs som synes i en terminal med namnet *tty1*. Dessa är i tur och ordning den virtuella terminalen (*init* kör även denna i enanvändarläge, i fleranvändarläge används istället normalt programmet **mingetty**) skalet (*/bin/sh*) och själva kommandot som listar processerna (*ps aux*).

De övriga processerna, med namn som börjar på *k* (keventd, kapmd, etc.) samt *bdflush*, *mdrecoveryd* är demoner som startats av kärnan, bara *minilogd* har startats av *init*. Den sistnämnda demonen, liksom *klogd* som används när systemet startas i fleranvändarläge, loggar händelser i kärnan till filen */var/log/messages* vilket är platsen där alla meddelanden från såväl kärnan som systemet i övrigt hamnar.[32]

De processer som startats av kärnan uppfattas kanske som en aning suspekta. De är nämligen *inte* startade av *init* (som de borde) men *ägs* ändå av *init*. Du kommer heller inte att finna någon programfil på din hårddisk med namnet *kswapd* t.ex. Vad som i själva verket kommer i dagern i dessa processer är det faktum att Linuxkärnan inte *är* en monolitisk kärna: den startar faktiskt en handfull vitala processer utanför själva kärnan. Linux är alltså snarast att betrakta som en hybrid mellan en mikrokärna och en monolitisk kärna.

Vilka de underliga kärndemonerna är varierar med versionen på kärnan och vilka moduler som kompilerats in i den. *kjournald* är dock för det mesta närvarande.

5.4.5 Dynamiska länkbibliotek

Dynamiska länkbibliotek är en slags så kallade *programbibliotek*. *Programbibliotek* i sin tur är komponenter som kan användas för att bygga *datorprogram*. Tanken är att på samma vis som en människa kan slå upp specifik kunskap i ett bibliotek skall också ett datorprogram kunna plocka ut specifik kunskap ur ett programbibliotek. Detta gör att det enskilda programmet inte behöver kunna ”allt” utan kan nöja sig med att anropa en kodsnutt som åstadkommer det som krävs. Detta lättar i praktiken arbetsbördan för en programmerare, eftersom det gör att denne inte behöver uppfinna hjulet på nytt för varje nytt program som skall, t.ex. använda en rad Internet-funktioner eller spela upp MP3-filer. Det gör att programmeraren kan fokusera på det som *just det här* programmet skall vara bra på.

[32] *minilogd* används bara i enanvändarläge, eftersom *klogd* normalt i sin tur använder *syslogd*-demonen, vilken inte startas i enanvändarläget.

Just programbibliotek är en av den fria mjukvarans stora fördelar: eftersom alla projekt är öppna har de kunnat enas om ett antal bibliotek som i princip alla använder. När det gäller proprietära mjukvaror är situationen tyvärr alltför ofta den precis motsatta. Denna höga grad av komponentisering är, som vi sett tidigare, helt naturlig för alla POSIX-system, och bidrar återigen till att fokusera resurserna på att göra en viss sak på *ett enda ställe* och att *göra det rätt* på detta ställe.

Vissa programbibliotek, som GNU C Library, är nödvändiga för systemets funktion. All kommunikation med operativsystemets kärna sker via detta och andra programbibliotek. Utan sådana bibliotek kommer programmen inte att fungera alls.

Dynamiska programbibliotek (dynamiska länkbibliotek, eller på engelska: *dynamically linked libraries*), laddas in i minnet och kopplas till ett program som använder det i samma stund som programmet körs. Detta sker naturligtvis av en anledning: det betyder nämligen att endast en kopia av programbiblioteket laddas in i datorns minne, oavsett hur många program som använder det. På så vis spar de dynamiska programbiblioteken minne, eftersom alla program som behöver ett visst bibliotek kan utnyttja ett och samma bibliotek på en och samma plats i datorns minne. Detta gör att fria mjukvaror, som använder fler sådana gemensamma bibliotek, är mer minnessnåla. När det gäller extremt välanvända bibliotek som GNU C Library är detta till och med något av en nödvändighet, annars skulle minnet snart ta slut.

Dessa bibliotek finns normalt i */lib* och */usr/lib* och har namn i stil med **libfoo.so**, **libfoo.so.1**, **libfoo.so.2** o.s.v.[33] Programmen som behöver länkbiblioteken kommer att leta efter dem där. Dessa namn, exklusive sökvägen, lagras i det program som använder det dynamiska länkbiblioteket, så om mitt program använder **libfoo.so.1** så lagras detta namn i mitt program under kompileringen. När programmet sedan körs kommer det att leta efter en fil med namnet **libfoo.so.1** för att länka sig till. Operativsystemet ser då till att fixa detta, och återanvänder också ett existerande programbibliotek i datorns minne om det redan är inladdat.

Eftersom programmet nu har blivit beroende av det dynamiska länkbiblioteket för att fungera kommer det att tvärstanna om det inte kan hitta det. I praktiken betyder det alltså, att om mitt program har använt **libfoo.so.1** och en fil med detta namn inte finns i */lib* eller */usr/lib* så kommer programmet inte att kunna starta, utan klagar istället vid prompten över att ett visst dynamiskt länkbibliotek saknas. Om du ar-

[33] Siffrorna efter *.so*-suffixet är en versionssiffra, som ändras när ny funktionalitet tillkomer i programbiblioteket, som är av sådan karaktär att det gamla inte fungerar på samma vis "utifrån sett", d.v.s. för de program som använder det.

betar i en grafisk miljö syns inte alltid sådana felmeddelanden, och du märker då bara att program som borde starta inte startas alls. Prova då att starta programmet från ett skal i en terminal och se om det rapporteras någon form av felmeddelanden.

Operativsystemet kommer dock att leta på *en* plats till: nämligen i de kataloger som pekas ut av miljövariabeln $LD_LIBRARY_PATH. Specifikt gäller det, att om du installerat programvara i */usr/local* så att programbibliotek har råkat hamna i */usr/local/lib* så måste du ibland skriva något i stil med **export LD_LIBRARY_PATH=/usr/local/lib** innan dina program fungerar. Detta löser dock bara problemet tillfälligt.

Om du vill lösa detta permanent är det inte helt säkert hur du skall gå tillväga, men i de flesta distributioner skall du editera filen */etc/ld.so.conf* och i denna lägga till sökvägar till alla kataloger som innehåller dynamiska länkbibliotek och som ligger utanför katalogerna */lib* och */usr/lib*.[34] Efter detta måste du oftast köra kommandot **ldconfig** för att det nya innehållet i den filen skall börja användas. Ett typisk problem i Red Hat och Fedora Core är att denna fil inte pekar ut */usr/local/lib* och att lokalt installerade programbibliotek därför inte kan hittas.[35]

Som du ser kan dynamiska länkbibliotek ge upphov till en del problem; problem som löser sig automatisk med distributionernas pakethanteringssystem, och detta är antagligen också en av de huvudsakliga anledningarna till att dessa paketeringssystem används. (Dessa problem är särskilt vanliga om du själv kompilerat olika program.)

Om du vill undersöka vilka dynamiska programbibliotek ett visst körbart program använder kan du skriva **ldd foo**. Detta kommer även att upplysa om ifall systemet kunde hitta de använda programbiblioteken eller inte. Testa t.ex. med **ldd /bin/ls**. Du kommer (föga förvånande) att upptäcka att de flesta program använder GNU C Library (som ofta heter *libc.so.6* eller något liknande).

Förutom dynamiska länkbibliotek finns det också *statiska programbibliotek*, vilket däremot aldrig vållar problem. Om de är installerade finns de också i */lib*, */usr/lib*, */usr/local/lib* o.s.v., men har filnamn som **libfoo.a**. När en programmerare använder dessa kopieras koden in i programmet vid kompilering av programmet istället för att länkas in när programmet körs. Det kompilerade programmet kan sedan kopieras till en helt annan maskin och köras där utan att det statiska programbiblioteket behöver finns där.

[34]Det är oftast också så att filen innehåller kommandot *include ld.so.conf.d/*.conf* vilket gör det möjligt att peka ut varje bibliotekskatalog med en egen fil som läggs i */etc/ld.so.conf.d*

[35]Red Hat och Fedora Core uppfattar själva inte detta som ett direkt fel, utan menar att det är så det *skall* fungera, eftersom en systemadministratör själv skall avgöra hur */usr/local* skall användas, men andra distributioner uppfattar denna ståndpunkt som felaktig.

5.4.6 Standardisering

Även om de olika distributionerna som beskrevs i kapitel 4 har det mesta gemensamt vad gäller innehållet av GNU/Linux-program, är verkligheten inte så vacker när det gäller driftsmiljön. De olika distributionerna har byggts upp var för sig och har flera utmärkande egenheter. Vissa av dessa egenheter, t.ex. valet av bootstrap loader och installationsprogram, har ingen skadlig inverkan på driftmiljön.

Andra saker är värre. Vissa distributioner har valt att inte använda *SysVinit*. Vissa har valt att inte lägga alla program i */bin* och */usr/bin*, en del har lagt vissa konfigurationsfiler i underkataloger till */etc*, och det inofficiella POSIX-filträdet på sidan 56 är inte något som alla GNU/Linux-system måste följa.

Ett annat skriande problem är att de dynamiska länkbibliotek som beskrevs i föregående avsnitt inte alltid finns tillgängliga.

Sammantaget leder detta till att ett program som är kompilerat och installerat på en dator med en viss GNU/Linux-installation, exempelvis en viss Red Hat Linux-installation, inte utan vidare kan installeras och köras på en annan dator med, säg, en Debian GNU/Linux-installation. Det *kan* fungera att bara kopiera den körbara programfilen från */usr/bin* till en annan dator, men är programmet någotsånär avancerat, är risken stor för att det inte fungerar alls.

Att alla som utvecklar GNU/Linux-program skall behöva paketera sina program för alla upptänkliga GNU/Linux-distributioner är omänskligt. De brukar därför ofta nöja sig med att tillhandahålla paket bara för ett par, och strunta i alla andra, eller uppmana dem att ”kompilera från källkod” (se kapitel 8). Denna situation gynnar de stora distributionerna på bekostnad av de små, och är därför inte acceptabel.

Av den här anledningen har *Linux Standard Base*[18] skapats gemensamt av medlemmarna i *Free Standards Group* där bland andra IBM, Hewlett-Packard, Intel, Red Hat, MandrakeSoft, SuSE AG, och Linus Torvalds arbetsgivare OSDL (Open Source Development Lab)[36] ingår.

Även om detta är en intresseorganisation för de ingående företagen har initiativet vunnit visst gillande, och exempelvis Debian GNU/Linux följer de av riktlinjerna som de anser vara meningsfulla. Linux Standard Base definierar bland annat:

- Att ett antal programbibliotek *skall* finnas tillgängliga. Dessa är bland annat GNU C Library (libc), GNU Math Library (libm), POSIX-trådar (libpthread) GNU C Compiler:s eget bibliotek, kryptofunktioner, C++-standardbiblioteket, ett autentiseringsbibliotek (libpam) o.s.v. o.s.v.

[36] I OSDL ingår sedan bl.a. IBM och Hewlett-Packard igen. . .

- Att ett paketeringssystem, som i praktiken är Red Hat:s RPM, fast något avskalat, skall användas. Detta gillas inte speciellt mycket av Debian-utvecklarna, men RPM-paket kan i någon mån numera användas även i Debian.

- Att ett antal kommandoradsverktyg skall finnas, varav de flesta är desamma som i POSIX-standarden, men några extra är tillagda, t.ex. kommandot **lsb_release** som förväntas rapportera vilken version av Linux Standard Base som stöds av operativsystemet.

- Att verktygen skall använda växlar på formen **kommando --växel** istället för kortformer, vilket skall förstås som att standarden bara specificerar beteendet med dubbla bindestreck — om du använder dina gamla hederliga enkelstreck som i **ls -a** så kommer beteendet att vara *odefinierat genetemot standarden* (även om det naturligtvis kommer att fungera bra, eftersom du med största sannolikhet kör GNU Fileutils **ls**-kommando).

  Förutom vad som sagts generellt om kommandon i avsnitt 2.3.1 gäller för kommandona i GNU/Linux-sviten att de ofta även accepterar en extra pratig variant av växlar, till exempel **kommando --password=fnord** i stället för den mer POSIX-lika syntaxen **kommando -p fnord** där alla växlar består av endast en bokstav.

- Att filsystemets struktur rekommenderas följa Filesystem Hierarchy Standard.

- Att körnivåer i stil med de som finns i *SysVinit* skall användas.

- Att ett antal miljövariabler för att ta reda på i vilket tillstånd maskinen befinner sig, och vad som finns anslutet, t.ex. nätverk.

Med hjälp av dessa standardfunktioner är det tänkt att det skall bli möjligt att tillverka ett enda RPM-paket av en mjukvara, som sedan skall kunna installeras på vilket GNU/Linux-system som helst, bara det följer Linux Standard Base.

Naturligtvis finns det en mängd GNU/Linux-system för vilka det vore helt meningslöst att följa Linux Standard Base. Specialsystem, som brandväggar, *X*-terminaler eller olika inbyggda system är exempel på sådana system.

5.4.7 Filsystemets hierarki

Filesystem Hierarchy Standard[10] är GNU/Linux-världens försök att standardisera det inofficiella filträdet från POSIX-filsystemen. Denna

standard är utformad så att det i princip skall kunna användas för andra operativsystem än GNU/Linux, och därför har de saker som är specifika för just GNU/Linux-system flyttats till ett eget appendix.

Filesystem Hierarchy Standard är avsedd för utvecklare, men är du nyfiken så skadar det aldrig att läsa den. Ett par specialiteter finns det dock i denna specifikation:

/proc skall innehålla ett pseudo-filsystem för processinformation. Numera använder Linuxkärnan detta till lite allt möjligt, såväl processer som information om maskinvaran och processorns temperatur lagras här. (Se vidare nästa avsnitt som handlar specifikt om */proc*.)

/sys är ytterligare ett pseudo-filsystem som tillkom med version 2.6.0 av Linuxkärnan. Detta har som syfte att visa en hierarkisk vy av hur olika hårdvaruenheter är anslutna till systemet. Delar av informationen i */sys* fanns tidigare i */proc* men flyttas nu successivt över till */sys*.

/usr/src är en plats där du kan lagra källkod till saker och ting. Mest är detta avsett för källkod till olika paket och till själva Linuxkärnan. Andra program som du kompilerar själv kan du lägga i */usr/local/src*.

5.4.8 Enhetsfilerna i /dev

De enhetsfiler som finns i */dev* i filsystemet och som presenterades på sidan 72 är i GNU/Linux nära kopplade till motsvarande modul i kärnan. Du kan undersöka dem genom att gå ned i katalogen */dev* och skriva **ls -al**. Detta kommer att ge en hel del information om blockenheterna, t.ex.:

```
crw-rw-rw-   1 root    root      1,  3 30 jan  2003 null
brw-rw----   1 sofie   floppy    2,  0 30 jan  2003 fd0
brw-rw----   1 root    disk      3,  0 30 jan  2003 /dev/hda
crw-rw----   1 root    uucp      4, 65 30 jan  2003 /dev/ttyS1
crw-------   1 sofie   root      5,  1 16 jan 02.45 /dev/console
```

Den första bokstaven längst till vänster som anger filtyp anger som väntat **c** (från engelskans *character*) för teckenenheter och **b** för blockenheter. Vi ser att användaren *sofie* är inloggad på datorns konsoll, eftersom hon äger */dev/console*-filen. Hon äger också av naturliga skäl flexskiveenheten, eftersom även denna sitter fastskruvad i konsollen.

Vi ser också att vissa grupper (*disk*, *floppy*, *uucp*) har definerats i systemet för att hålla reda på vilka användare som har rätt att komma åt

olika enheter. Filrättigheterna ger ju i sin tur tillgång till filen för vissa grupper av användare, och eftersom enheten representeras av en fil så kommer filrättigheterna att styra tillgången till enheten.

De två siffrorna som följer direkt efter gruppnamnet, på den plats där fillängden brukar stå för en vanlig fil, kallas *större enhetsnummer* respektive *mindre enhetsnummer*. Dessa nummer visar hur Linuxkärnan kopplar ihop en enhetsfil med gränssnittet på en modul i kärnan: nummer 1, 2, 3, 4 och 5 i detta exempel är motsvarande gränssnitt mot kärnan. Eftersom enheter i exemplet är viktiga och så att säga grundläggande för systemet, så har de låga nummer. De mest rudimentära, som */dev/null* och */dev/zero* har nummer 1, sedan numreras de uppåt. Modulen som har hand om diskettstationen använder nummer två, o.s.v.

Det *mindre enhetsnumret* är ett löpnummer. För ATA/IDE-hårddiskar har exempelvis */dev/hda* löpnummer 0, medan partitionerna på hårddisken */dev/hda1*, *hda2*, *hda3* etc. får nummer 1, 2, 3 o.s.v. Nästa ATA-hårddisk */dev/hdb* använder samma gränssnitt men börjar numreringen för löpnumret på 64. (Detta är inte det enda skälet till att du skall undvika att ha fler än 63 partitioner på din hårddisk, men så gott som något.)

En drivrutin i kärnan talar om vilket större enhetsnummer den avser att använda (om den vill kommunicera med operativsystemet och användaren på detta vis) och får sig sedan tilldelat ett mindre enhetsnummer beroende på i vilken ordning de olika drivrutinerna laddats in i kärnan och aktiverats. När ett program i systemet sedan öppnar motsvarande enhetsfil kommer tecken som läses eller skrives till denna att skickas till drivrutinen för bearbetning. Drivrutinen kommer därefter att sköta hanteringen av hårdvaran (eller annan funktion) som den har till uppgift att handha.

Det finns också ett antal gränssnitt som har tilldelade gränssnittsnummer, men som inte finns i */dev*-delen av filsystemet. Dit hör framför allt nätverksgränssnitten *eth0*, *ppp0*, etc.

Det finns två olika sätt att hantera */dev*-hierarkin. Det gamla sättet, som förekommer i många distributioner, innebär att denna del av filsystemet fylls med i princip alla tänkbara enheter som över huvud taget kan komma att dyka upp. Det innebär att om du skriver **ls /dev** så ser du en lista på hundratals eller tusentals tecken- och blockenhetsfiler som till största delen är helt oanvändbara, eftersom du inte har dem. Om någon av dessa saknas kan det hända att du blir tvungen att skapa dem själv. Du måste då veta vilket namn som din enhetsfil skall ha, t.ex. *foo0*, och skapar den då med:

```
cd /dev
./MAKEDEV foo0
```

Som synes ligger det ett program med namnet **MAKEDEV** med stora bokstäver i */dev*, som bara är avsett för att skapa nya enhetsfiler. Om du inte vet vad din enhet skall heta, eller har skrivit en helt ny drivrutin eller något ditåt, kan du använda programmet **mknod**, förutsatt att du vet vilket större och mindre enhetsnummer drivrutinen använder. Om din drivrutin är en teckenenhet och har större enhetsnummer 42 och mindre enhetsnummer 5, skapar du enhetsfilen *foo* med:

```
cd /dev
mknod foo c 42 5
```

Detta system med mycket stora mängder underliga enhetsfiler fasas successivt ut till förmån för ett system som dynamiskt skapar sådana filer när enheter detekteras av Linuxkärnan. Detta system går under namnet *udev* (detta namn betyder inget speciellt). Om din installation av Linux använder *udev* kommer programmet */sbin/udev* att köras varje gång en ny enhet detekteras, varefter en uppsättning regler i */etc/udev/rules.d* tillämpas för att korrekt namnge motsvarande block- eller teckenenhet.[37] Du kan direkt se detta på att det är förhållandevis få filer i */dev*. Vissa distributioner använder en blandning av det gamla och det nya systemet, så att det är ganska många filer i */dev* men större delen ändå hanteras på det nya sättet.

Det nyare systemet med *udev* ger även användare möjligheten att lägga till egna regler så att en viss enhetsfil har ett konsistent namn, t.ex. att en specifik kamera kan dyka upp som */dev/hasselblad* varje gång den ansluts, snarare än som ett numrerat löpnummer under */dev/sdaN*. Att skriva sådana regler är emellertid något komplicerat.

5.4.9 Kärn- och processkontroll — /proc

/proc är som en öppning in i Linuxkärnans maskinrum. Det är en katalog med filer och underkataloger som inte är riktiga filer eller kataloger. Alla filer har storleken noll, och innehållet i kataloger och filer ändrar sig hela tiden för att återge systemets tillstånd.[38] Ursprungligen var */proc* avsett för att ge en listning över processer som körs i kärnan, men har nu blivit mycket mer.

Alla filer som ligger i */proc* är textfiler, så du kan titta på dem med **cat foo** eller **less foo**, men formatet är ibland kryptiskt, eftersom det är meningen att ett program skall läsa av informationen.

[37] Dessa regelfiler namnges och tillämpas i en ordning som är identisk med uppstartfilerna för init-processens skript för olika körnivåer i */etc/rc.N*, se avsnitt 5.4.3 på sidan 189.

[38] I själva verket är det naturligtvis inte så att innehållet *ändras*, de *genereras* helt enkelt när en användare försöker titta på dem.

Om du går in i katalogen */proc* kan du se två saker: först ett antal kataloger med siffernamn. Dessa kataloger motsvarar de processer som kör på din dator just nu. Om du skriver **ps** vet du vilka processnummer som hör till just ditt skal,[39] och på så vis kan du undersöka "dina" processer närmare. Om du går in i katalogen med ditt processnummer, kan du se en hel del spännande och en del mindre spännande saker:

cmdline är en fil som innehåller namnet på programmet som startade denna process, d.v.s. vad som skrevs i skalet, om du startat processen själv. Om du inte startat processen själv, så är det namnet på den programfil som systemet har anropat för att starta processen.

cwd (från engelskans *current working directory*) är en länk till processens arbetskatalog. Om du utför kommandot **cd** till denna länk hamnar du i processens arbetskatalog, någonstans i filsystemet. Om du har läst C.S. Lewis bok *Min morbror trollkarlen* minns du kanske att huvudpersonerna kommer till en skog med olika dammar som öppnar dörrar mellan olika världar. *cwd* är som dammarna i skogen: du kan ta dig in i olika processers "världar".

environ är en lista över miljövariabler och de värden de hade i skalet då processen startades. Denna lista är avdelad med NULL-tecken[40] och ser inte riktigt frisk ut ifall du tittar på den med t.ex. **cat**, prova med **cat environ | tr "\0" "\n"** (det var inte jag som kom på de larviga NULL-tecknen!)

exe är en länk i filsystemet till den exekverbara fil som startade denna process.

maps talar om vart i minnet de olika delarna av processen har hamnat, i synnerhet var de olika dynamiska länkbiblioteken har laddats in.

status ger information om processens tillstånd och hur mycket minne den tar upp totalt. Det finns även en fil med namnet **statm** som ger samma typ av information fast i mera kryptisk form.

Andra delar av */proc* ger annan information. Några filer ligger direkt i */proc*, och innehåller en hel del information som är relativt obegriplig för den som inte kan en del om IBM PC-kompatibel hårdvara, här är *några* av dem:

[39] I själva verket hämtar själva kommandot **ps** också sin information från */proc*.

[40] NULL är tecknet med teckenvärde 0 (noll)

/proc/cpuinfo talar om allt som är värt att veta om processorn i din dator, som vilket märke och vilken revision det är, om det finns några speciella egenskaper eller fel med den (tidiga versioner av Intels Pentium-processor hade inbyggda fel som måste undvikas), hur stort cache-minne den har, vilken hastighet den körs på o.s.v. Om du vill veta hur snabb en okänd dator är, och vilken typ av processor som sitter i den, så är det denna fil du skall titta i.

/proc/devices talar om vilka enheter (teckenenheter och blockenheter) som är anslutna till systemet. Dessa finns också normalt i */dev* i en eller annan form, men här representeras de av en siffra.

/proc/dma ger information om vilka DMA-kanaler som är uppsatta för datorns buss.

/proc/filesystems ger information om vilka filsystemsmoduler som är installerade i kärnan, och om de är anslutna till någon enhet. (Se mer i avsnitt 5.4.10.)

/proc/interrupts ger upplysningar om vilka avbrott som konfigurerats i maskinen, och vad de används till.

/proc/ioports talar om vilka *hårdvaruportar* som konfigurerats i datorn. Portarna som avses är *inte* detsamma som TCP/IP-portar eller garageportar, utan minnesområden i datorns första 64 kilobyte minne som kan användas av olika instickskort och liknande. Att skriva eller läsa till dessa portar är den huvudsakliga kommunikationsvägen för kärnan när den vill kontrollera datorns hårdvara.

/proc/meminfo ger information om hur minnet i din dator används.

/proc/modules ger samma information om vilka moduler som laddats in i kärnan som kommandot **lsmod**.

/proc/pci ger uppgifter om vilken hårdvara som är ansluten till datorns PCI-buss. Normalt kör du dock kommandot **lspci** för att ta reda på detta, eftersom det senare ger en mer kompakt och lättläst utskrift.

/proc/swaps talar om vilken växlingsenhet (eller vilk*a* växlingsenhet*er*) som används för virtuellt minne.

I */proc* kan du även hitta underkataloger med information för SCSI-enheter, USB, o.s.v. I princip alla delar av kärnan som du väljer att kompilera in eller ladda in i form av moduler kan skapa en eller flera filer eller kataloger här.

En viktig katalog är */proc/sys* där alla *parametrar* i kärnan finns tillgängliga och kan både undersökas och ändras. (De andra delarna av */proc* är skrivskyddade och kan inte ändras.) De flesta parametrar som finns här ställs in under uppstart av systemet, och vill du ändra dem finns kommandot **sysctl** som kan utföra ändringar i */proc/sys* på ett strukturerat vis (se nedan). Till exempel kanske du vill ändra antalet maximalt samtidigt öppna filer i systemet: detta lagras i */proc/sys/fs/file-max*.

För att underlätta för användare att få ut information från det ganska kryptiska formatet i */proc* (och via direkta anrop till Linuxkärnan) har tre programpaket utvecklats: Psmisc, Procinfo och Procps. Dessa är inte alltid installerade i alla distributioner, och då kan det krävas att du först installerar dem för hand. Eftersom de inte är livsnödvändiga hamnar de normalt i katalogen */usr/sbin*.

Psmisc är relaterat till enbart processer och processkontroll och innehåller tre viktiga och bra kommandon:

killall programnamn dödar alla processer som är instanser av ett visst program. Om du exempelvis har startat sju olika EMACS-editorer, så kommer kommandot **killall emacs** att avsluta dem alla. Observera att om du kör detta kommando som root-användare, så kommer alla andra användares eventuella EMACS-sessioner också att avslutas, vilket kanske inte var vad du önskade. Liksom det vanliga **kill**-kommandot kan en viss signal anges, t.ex. **killall -9 emacs**.

Förväxla *aldrig* detta kommando med kommandot **killall5**, som är en implementation av ett UNIX System V-liknande killall-kommando: detta dödar *alla* processer, så om du råkar vara root kommer hela ditt system att stanna. I annat fall blir du bara utslängd. Detta kommando skall bara användas av *init* när det är dags att stänga ner systemet.

pstree ger en grafisk trädliknande bild som visar hur de olika processerna är släkt med varandra i förälder-barn-relation. Urföräldern är alltid processen *init*.

fuser foo.txt tar reda på vilken process som använder filen *foo.txt* just nu, om någon process gör det — i annat fall returneras bara en tomrad.

Procinfo innehåller program avsedda för att få ut information om hårdvaran i systemet:

lsdev ger information om vilka hårdvaruenheter som är anslutna till vilka portar.

procinfo (med samma namn som programpaketet) ger information om systemets allmänna hälsa: hur mycket minne som är använt (både fysiskt och virtuellt), när systemet startades och en uppskattning om belastningen fram till nu, samt vilka avbrott som är konfigurerade och vilka uppgifter de har.

socklist ger uppgift om vilka TCP/IP-socklar som är öppna på systemet, och vilka processer (ofta demoner) som använder dem. Detta kommando är mycket bra för att hålla ett öga på nätverkssäkerheten, men det vanligaste kommandot för detta ändamål är **netstat** som ger en ännu trivsammare sammanställning. (Läs mer i kapitel 9 om nätverket.)

Procps-paketet är specifikt avsett för att plocka fram information som berör systemets processer, men viss funktionalitet överlappar Procinfo:

free talar om hur mycket minne som är använt, och hur mycket som är ledigt. Det minne som kallas *swap* är detsamma som det virtuella minnet.

Du skall inte drabbas av panik om det ser ut som att nästan hela minnet är uppätet av olika processer; Linux är skrivet för att utnyttja minnet maximalt om det går, och kommer därför att lägga in så mycket efterfrågad data som är fysiskt möjligt i minnet, t.ex. filer som begärts från hårddisken många gånger. Om det fysiska arbetsminnet behövs till något annat kommer onödiga områden att frigöras eller kopieras ut till virtuellt minne.

Det verkliga hotet mot minnet uppstår då alla de processer som samtidigt körs i datorn inte får plats i minnet: då kommer Linux att försöka skyffla hela processer ut- och in ur det virtuella minnet, vilket manifesterar sig i en hysterisk diskaktivitet: systemet *harvar*. Du brukar kunna höra på din hårddisk om detta inträffar: det knattrar något alldeles förfärligt. Detta fenomen är inte ovanligt på datorer med litet fysiskt minne!

pgrep är ett smidigt program för att lista processnummer enligt vissa villkor, som vilket namn de har eller vilken användare som startat processen. Har vissa likheter med POSIX-kommandot **ps** och är väl till viss del överflödigt.

pkill kan döda en process med utgångspunkt från namnet på den programfil som startade processen. Likheter med **killall** och POSIX **kill**, men återigen med vissa mindre variationer. POSIX **kill** kan

inte döda processer med utgångspunkt från programfilernas namn, till exempel.

pmap PID ger information om exakt vilka minnesareor som används av en viss process, med utgångspunkt från dess processnummer. (Hämtas relativt oförändrat från filen */proc/<PID>/maps*.)

skill är ytterligare ett kommando som gör liknande saker som **kill**, **killall** och **pkill** redan gör. Programmet har däremot inget med skicklighet att göra.

snice har likheter med POSIX **nice**, men kan välja processer efter namn eller användare, t.ex.:

- ”prioritera ned alla processer som tillhör användaren *bertil*”: **snice +2 -u bertil**, eller
- ”prioritera ned Mozilla”: **snice +7 -c mozilla-bin**.

Växlarna här, **-u** och **-c** kan oftast uteslutas, eftersom **snice** är listigt nog att lista ut om det rör sig om en användare eller ett namn på en programfil som startat en process, helt på egen hand.

sysctl kan modifiera parametrar till kärnan, d.v.s. värden i */proc/sys* under drift. Du *kan* ju såklart gå in i */proc/sys* och ändra värdena direkt i /proc-filsystemet, men vill du det?[41] Exempelvis kan du ändra namnet på din dator (som under uppstart sätts till ett visst värde) med **sysctl -w kernel.hostname=foobar**, och iaktta resultatet med **sysctl -n kernel.hostname**.

För att ha en aning om vad du kanske vill skruva på får du såklart undersöka katalogen */proc/sys*. Parametrarna till **sysctl** är som du ser på formen *katalog.nyckel*, där *katalog* är en underkatalog till */proc/sys*.

tload ger en graf som propagerar från vänster till höger och som visar systemets belastningsgrad med grovkornig grafik.

top är ett populärt kommando som dessutom finns i de flesta POSIX-system. Detta ger en lista över processer som kör just nu, ordnad efter hur mycket av processorns kapacitet de tar upp, d.v.s. en ”topplista”. Så fort du tycker att din dator känns seg, och undrar ”vad gör den nu?” är det dags att köra **top**.

[41] Folk som vill briljera kan ibland få för sig att ändra en kärnparameter med ett kommando av typen **echo ”foobar” > /proc/sys/kernel/hostname**, vilket inte är någon större poäng gentemot **sysctl**.

Kommandot körs så att listan uppdateras var 5:e sekund. Om du vill köra kommandot med maximal uppdateringsfrekvens kan du skriva **top q**. Detta gör att systemet tar all ledig tid som finns för att visa processinformation.

Om det inte är processorns belastning som är flaskhalsen för tillfället kan **top** även visa en topplista ordnad efter hur mycket minne som processerna använder o.dyl. genom att du, medan **top** körs, trycker på någon av tangenterna:

P (d.v.s. håll nere SHIFT-tangenten och tryck på tangenten P) sorterar efter hur mycket av processorn som processerna använder just nu. (Standardläge.)

M sorterar processerna efter hur mycket minne de använder.

N sorterar efter process-ID-nummer.

A sorterar processerna efter hur gamla de är.

T sorterar processerna efter hur mycket av processorn de använt *totalt*, d.v.s. ”de tyngsta programmen över tid”. Detta inkluderar naturligtvis bara program som fortfarande körs. Demoner och processer som alltid är igång (t.ex. fönstersystemet *X*) hamnar ofta högt upp.

Listan från **top** uppdateras kontinuerligt i ditt terminalfönster så att du kan se aktiviteten i ditt system utvecklas successivt. Avsluta programmet genom att trycka på tangenten **q** (quit).

uptime är ett simpelt kommando som talar om hur länge systemet varit i drift sedan det senast startades. (Somliga tävlar med varandra om att ha längst uptime, och undviker i det längsta att starta om datorn.)

hdparm är ett separat program som används för att konfigurera hårddiskar, CD-ROM-enheter och andra enheter som använder IDE/ATA-bussen.[42] Hdparm kan skruva på alla möjliga parametrar för dessa enheter, men det problem som användare normalt använder hdparm för att lösa är *långsamhet*.

För att åtgärda långsamhet med hdparm brukar kommandot **hdparm -d1 /dev/foo** lösa problemet. Om du är mera risktagande kan du prova **hdparm -d1 -X33 /dev/foo**. Detta är ibland nödvändingt för att somliga DVD-läsenheter skall kunna spela upp en film i normalt tempo!

[42] Det egentliga namnet på denna buss är ATA (Advanced Technology Attachment) men de flesta använder ändå den gamla beteckningen IDE (Integrated Drive Electronics).

(**hdparm -d1 -X33 /dev/dvd**, eller **/dev/cdrom**.) Även vanliga hårddiskar kan vara långsamma, men då är det desto farligare med sådana experiment, eftersom dessa också ska skriva till disken, och då kan saker gå ordentligt fel om en felaktig parameter angivits, detta gäller i synnerhet **-XNN**-parametern.[43]

Om du vill veta mer om */proc* och hur detta filsystem används, finns ett utförligt dokument i Linuxkärnans källkodsträd skrivet av utvecklare vid SuSE, se [5].

5.4.10 Filsystemstyper

Innan vi säger något om filsystemtyper i Linuxkärnan måste vi specificera vad vi menar med detta ord. Ordet *filsystem* har nämligen i GNU/Linux-system två skilda betydelser:

Betydelse 1: det system/hierarki av *filer* som bildas av kataloger, vanliga filer, länkar, symboliska länkar, block- och teckenenheter o.s.v. (Se avsnitt 2.3.5 på sidan 51.)

Betydelse 2: det system som används för att organisera filernas innehåll på ett visst lagringsmedia såsom den magnetiska ytan på hårddiskar, disketter o.s.v.

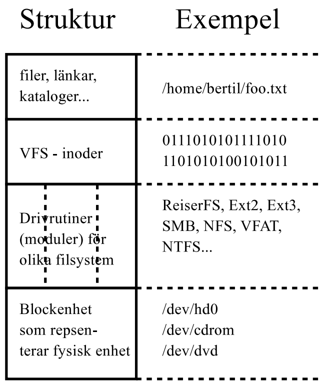
Den del av Linuxkärnan som har hand om båda dessa delar kallas *Virtual Filesystem*, det virtuella filsystemet eller kort och gott VFS.[44] I källkodsträdet för Linux heter samma del **fs**. Principen för VFS illustreras i figur 5.7.

Vi vet att en mängd metadata skall lagras tillsammans med filen, såsom ägare, tid då den skapades etc. Detta är filsystemet i betydelse nr 1. Sådan information lagras av kärnan i strukturerad form i *inoder*, och själva datan i filen, text, ljud eller vad det nu är, lagras också i inoder. Inodernas struktur tillhandahålls av VFS. VFS skickar sedan denna struktur bestående av enbart inoder vidare till en modul i Linuxkärnan som har hand om ett filsystem i betydelse nr 2 och organiserar dessa inoder på ett listigt vis på någon blockenhet.

Blockenheten där ettorna och nollorna till sist hamnar kan vara det magnetiska skiktet på en hårddisk, men också någon disk som anslutits över nätverket, en USB-nyckelring eller liknande saker som kan lagra data som en lång bitvektor. (Hålremsor eller DAT-band fungerar utmärkt, men är ganska långsamt.)

[43] För mer information om dessa magiska växlar, läs manualsidan för hdparm.

[44] För att ytterligare förvirra användaren har skrivbordshanteraren GNOME *också* en filsystemliknande komponent, som *också* heter VFS. (KDE:s motsvarighet heter IOSlaves.) Förväxla inte. . . Ja det är kanske inte så lätt.



Figur 5.7: Principen för det virtuella filsystemet, VFS. De filer användaren upplever sig manipulera omvandlas till s.k. *inoder*, vilka är en ström av strukturerad data. Denna data struktureras än en gång i ett av flera möjliga filsystem, som i sin tur ansluter till en viss blockenhetsfil.

Linux virtuella filsystem tillåter att flera olika filsystem av typen 2 används parallellt, d.v.s. du kan t.ex. ha *både* Linux-, Windows- och Macintosh-filsystem monterade i ett och samma filträd. För en användare är de alla ett och samma filträd.

Filsystem för hårddiskar

Linuxkärnan har stöd för flera alternativa system när det gäller att organisera filerna på en fysisk enhet. Det första filsystemet för hårddiskar var *Minix* eget filsystem, eftersom det var på ett sådant som kärnan ursprungligen utvecklades. Detta filsystem hade stora begränsningar (t.ex. kunde det bara hantera blockenheter med en storlek upp till 64 megabyte), och därför utvecklades redan år 1992 ett *utökat filsystem* eller *ExtFS*, som det kom att heta. Inte heller detta var särskilt fullödigt, och därför utvecklade Rémy Card, Theodore Ts'o och Stephen Tweedie under 1993 ett andra utökat filsystem, *Ext2*. Detta är alltjämt det vanligaste och förmodligen snabbaste filsystemet i Linuxkärnan.

Problemet med Ext2 var att det inte stödde journalföring, vilket framför allt är viktigt att ha om något fel uppstår på en hårddisk under drift eftersom tiden det tar att gå igenom en "smutsig" hårddisk (som stannat under drift) är proportionell mot diskens storlek eftersom hela utrymmet måste sökas igenom. Om ett Ext2-filsystem stannar under drift kan det ta flera timmar att åter ta det i drift om en modern, stor hårddisk används. När hårddiskarna var små var detta inget stort problem,

Vanliga filsystem

Kortnamn	**Namn**
ext2	Second extended filesystem
ext3	Journalling second extended filesystem
reiserfs	ReiserFS
jfs	IBM Journalled Filesystem
xfs	SGI IRIX Filesystem
minix	Minix filesystem
romfs	ROM Filesystem
isofs	CD-ROM filsystem, High Sierra File System med Rockridge extensions, ett CD-ROM-skiveformat som används av flera POSIX-system
udf	DVD-ROM filsystem
msdos	MS-DOS (riktigt gamla versioner)
vfat	Utökat MS-DOS filsystem med långa filnamn (introducerat med Windows 95)
ntfs	New Technology Filesystem, används av Windows NT, Windows 2000 och Windows XP
hfs	Apple Macintosh-filsystem
proc	används av /proc-filsystemet

Nätverksfilsystem

Kortnamn	**Namn**
nfs	Network File System (Sharing) protocol, standardfilsystem för nätverk i POSIX-miljö
smbfs	Server Message Block, Microsoft Windows nätverksfilsystem
cifs	Avancerade delar av Microsofts utökningar av Server Message Block
ncpfs	Novell NetWare Core Protocol-filsystem

Figur 5.8: Några vanliga filsystem som stödjs av Linuxkärnan. Det finns fler.

men ju större diskarna blev, desto större blev också problemet med att behöva vänta medan disken kontrollerades. Ett annat problem var att Ext2 skalade dåligt: i system med hundratals diskar och terabyte av data organiserade enligt RAID[45]-principen började Ext2 att bete sig riktigt dåligt.

Flera skribenter inom Linuxvärlden påtalade att denna brist kunde hindra Linux från att användas i stor skala, och därför infördes i senare versioner av Linuxkärnan (2.4-serien samt version 2.6) en hel skog av journalförande filsystem: *Ext3* är en utökning av Ext2 som kan hantera journalföring, *ReiserFS* är ett filsystem utvecklat av Hans Reiser, en person som redan tidigt gick ut med mycket bestämda åsikter om hur filsystem skulle skrivas. JFS är ett filsystem utvecklat av IBM och XFS är Silicon Graphics idé om hur det hela skall skötas.

Om du installerar GNU/Linux hemma på en skrivbordsdator är det lite hugget som stucket vilket filsystem du väljer. Jag skulle personligen i dagsläget välja Ext3 eller ReiserFS. Det finns även en begränsad möjlighet att använda Microsoft Windows filsystem NTFS om detta önskas, men denna modul är främst avsedd för att montera och läsa och skriva till Windows-filsystem om det skulle behövas. Denna modul är i dagsläget inte speciellt välutvecklad och kan t.ex. i skrivläget inte skapa nya filer i NTFS-system, bara modifiera befintliga filer.

[45] Se sidan 421

Filsystem för nätverk

Utöver de filsystem som är avsedda att användas för att lagra information på hårddiskar fastskruvade i den fysiska dator där Linuxkärnan körs, finns också ett antal *nätverksfilsystem* tillgängliga i kärnan. Dessa kräver att en annan dator tillhandahåller lagringsutrymme för filer, och de skickas sedan via nätverket till den datorn.

Eftersom det handlar om datorer i nätverk tillämpas klient-serverterminologi på dessa system. Datorn som tillhandahåller filsystemet kallas *filserver*, medan de datorer som ansluter sig till den kort och gott kallas *klienter*.

Följande nätverksfilsystem stödjs direkt av Linuxkärnan:

NFS eller **Network File System** (ibland även kallat **Network File Sharing**) är det mest ursprungliga nätverksfilsystemet i POSIX-världen. Det uppfanns av Sun Microsystems och använder undertill nätverksprotokollet UDP, inte TCP, som så många andra protokoll. (Den nyare versionen NFSv4 har dock stöd för TCP.) NFS är standardiserat av IETF (Internet Engineering Task Force) i form av RFC 1813 och RFC 1310. (Se sidan 300 om nätverksprotokoll vid behov.) Version 2.6.0 av Linuxkärnan (och högre) har stöd för NFSv4, en förbättrad variant av NFS-protokollet.[46]

SMB eller **Server Message Block** är det nätverksprotokoll som Microsoft Windows och även den fria filservern Samba använder. Vissa avancerade delar av detta protokoll kallas i Linuxvärlden för CIFS (Common Internet Filesystem), men det är också Microsofts eget namn på SMB och allt vad det innebär, så det är upplagt för förvirring. (Se vidare sidan 413.)

NCP eller **NetWare Core Protocol** är Novells eget nätverksfilsystemprotokoll.

Utöver dessa finns ett experimentellt stöd för andra, udda men lovande nätverksfilsystem, t.ex. AFS (Andrew Filesystem), Coda, Intermezzo m.fl. Appletalk, som används av en del Macintosh-datorer, stöds också.

Filsystemkommandon

För att hantera filsystemen finns en del oundgängliga kommandon. Dessa inkluderar inte kommandon för att hantera filer på det vis som

[46]Se vidare avsnitt B.2 på sidan 420. Många har starka åsikter om kvalitén, eller snarare bristen på kvalitet, hos NFS. Rich Sladkey som var den förste att implementera NFS i Linuxkärnan kallar det ”ett uråldrigt och hjärnskadat protokoll” (min översättning)[21]

en användare gör (t.ex. **ls** och **mv**) eller de kommandon som partitionerar hårddiskar (som **fdisk**) utan är specifikt inriktade på just *filsystemen*. De flesta av dessa kommandon kräver att du agerar som root:

mount foo bar används för att montera blockenheten *foo* på platsen *bar* i filsystemet. Exempel:

```
mount /dev/hdb /mnt/foo
```

Kommandot monterar hela hårddisken **hdb** som */mnt/foo*. Katalogen */mnt/foo* måste redan finnas (och om det finns andra filer eller kataloger där kommer de helt sonika att *gömmas*, d.v.s. de blir osynliga i filsystemet) och blockenheten */dev/hdb måste* innehålla ett fungerande filsystem av en typ som Linuxkärnan kan hantera. Ett annat exempel:

```
mount /dev/dvd /mnt/cdrom
```

Detta monterar blockenheten */dev/dvd* som normalt skall vara en DVD-läsare, på monteringspunkten (= katalogen) */mnt/cdrom*. Återigen måste det verkligen *finnas* en CD-ROM- eller DVD-skiva i DVD-läsaren, den måste ha ett acceptabelt filsystem, och katalogen där den skall monteras (monteringspunkten) *måste* finnas, men får gärna vara tom.

Om du vill ange att en speciell filsystemtyp används på den enhet som pekas ut kan du skriva t.ex. **mount -t vfat /dev/hda2 /mnt/dos** för att montera andra partitionen på första IDE-hårddisken i monteringspunkten */mnt/dos*, om detta har filsystemtypen VFAT, vilket är Microsoft Windows 95 standardfilsystem. Tillvägagångssättet är utomordentligt om du har mer än ett operativsystem på din hårddisk och vill kunna komma åt filerna i det andra operativsystemet från GNU/Linux. Om du vill ha flera ”egna” monteringspunkter av det här slaget är det bara att skapa dem med **mkdir** i */mnt*, detta ställe är reserverat för just sådant.

Mount har en konfigurationsfil i */etc/fstab*, ”filsystemtabellen”, där inställningar för olika partitioner bör lagras, om avsikten är att de skall monteras mer än en gång. Raderna i */etc/fstab* innehåller sex olika parametrar avskilda med mellanslag, och har följande innebörd, t.ex:

```
/dev/hdb  /mnt/data  ext3  defaults  0  0
```

Detta betyder att blockenheten */dev/hdb* skall monteras på monteringspunkten */mnt/data*, att den innehåller ett filsystem av typen Ext3, och att den skall monteras med standardflaggor (det finns många olika flaggor, se **man mount** eller **man fstab**). De två sista siffrorna betyder att enheten inte skall dumpas med kommandot **dump** respektive att filsystemet inte behöver kontrolleras med **fsck** vid uppstart.

Med hjälp av */etc/fstab*-filen kan mount på egen hand avgöra hur ett visst filsystem skall monteras, så om ovanstående rad är inlagd räcker det med att skriva **mount /dev/hdb** *eller* **mount /mnt/data** för att montera den här enheten i */mnt/data*.

När systemet startas upp kommer alla filsystem som är uppradade i */etc/fstab* att monteras, om inte parametern *noauto* angivits (som t.ex. för CD-ROM-läsaren). Har du ingen aning om vilket filsystem som kommer att användas för en viss enhet kan du skriva *auto* som filsystem, så kommer mount att testa sig fram till ett som (förhoppningsvis) fungerar.

Under drift skapar mount också filen */etc/mtab*, "monteringstabellen" där du kan se vilka filsystem som är monterade just nu, samt exakt vart de har monterats. Denna fil är i princip identisk med */proc/mounts*, och skiljer sig bara i fråga om hur de visar monteringen av rotfilsystemet. Kommandot **df** visar också monterade filsystem, och dessutom fyllnadsgrad.

Även nätverksfilsystem kan monteras:

```
mount -t nfs -o rw 192.168.1.2:/export/foo /mnt/foo
```

Kommer att montera det exporterade NFS-filsystemet på datorn 192.168.1.2 med namnet */export/foo* i din dators */mnt/foo*-katalog, i läs/skrivbart läge. Du kan även skriva in en liknande rad i */etc/fstab* om du vill att detta skall monteras automatiskt när datorn startas.

```
mount -a
```

Söker igenom hela */etc/fstab*-filen och monterar alla monteringspunkter som inte redan är monterade. Detta görs av systemet vid uppstart, men om t.ex. en NFS-server inte är tillgänglig vid uppstart kommer punkterna på denna inte att monteras, och root kan vara tvungen att gå in och göra det manuellt med detta kommando.

umount foo av engelskans *unmount* utför omvändningen till **mount**: kommandot monterar av en monteringspunkt som redan monterats, t.ex. kan användarnas hemkataloger avmonteras med kommandot **umount /home** om dessa lagts på en egen partition eller separat hårddisk.

swapon slår på det speciella filsystem som används för virtuellt minne, eller ”swap” som det ju också kallas. Partitionen för detta specialfilsystem hittar kommandot i */etc/fstab* på en rad där både monteringspunkt och filsystem har namnet *swap*.

swapoff stänger av växlingspartitionen för virtuellt minne. Detta är inte bra att göra om du inte vet vad du sysslar med. Både **swapon** och **swapoff** sköts normalt av *init*-processen under uppstart och nedstängning av systemet.

fsck foo (utläses *file system check*) där *foo* är antingen en blockenhet (t.ex. */dev/hdb*) eller en monteringspunkt (t.ex. */mnt/data*) kommer att undersöka om filsystemet som finns där är välmående, och kommer även vid behov att erbjuda sig att reparera eventuella fel. För vissa filsystemstyper går det snabbt att köra fsck, för andra tar det en evighet.

Om filsystemet på en viss enhet mår bra gör fsck ingenting. Hittas fel erbjuds du att få dem rättade, och du skall då normalt bara svara ”ja” (**y**es). Programmet kan även hitta filer som blivit ”hängande i luften”, d.v.s. som inte sitter fast någonstans, men som ändå finns på hårddisken. Det erbjuder dig då att ansluta dessa till katalogen *lost+found* i filsystemet på varje partition. (Den vanligaste platsen är i roten: */lost+found*.) Det är ytterst sällan du kan återskapa någon vettig information i *lost+found*, men det känns kanske tryggt att det åtminstone är teoretiskt möjligt att hitta sin fil där om t.ex. datorn stannat p.g.a. ett strömavbrott.

fsck kommer att anropa olika program beroende på vilket filsystem som du vill kontrollera. Varje filsystemstyp som stödjs av Linuxkärnan har ett eget sådant program.

Om du vill köra fsck på ett filsystem bör det först vara monterat i *skrivskyddat läge*. Detta innebär att det är skrivskyddat så att vanliga användare inte kan göra några ändringar i filsystemet med ”vanliga” kommandon (**mv**, **rm**...) medan root däremot kan utföra ändringar på själva filsystemet. Detta görs för att undvika att kontroll eller reparationer görs på ett system medan det ändrar sig, vilket är svårhanterligt. Ett filsystem kan monteras om i skrivskyddat (engelska: read-only) läge med **mount -o remount,ro foo**.

Efter att du kört fsck kan du montera om det i skrivbart läge igen med **mount -o remount,rw foo**.

mkfs -t foo /dev/bar skapar ett nytt filsystem av typen *foo* på blockenheten *bar*. Det går även bra att ange ett namn på en monteringspunkt — under förutsättning att denna finns med i */etc/fstab*.

Du skall normalt inte ge dig på att skapa ett filsystem där det redan finns. Det normala användningsområdet för detta kommando är då du köpt en ny, präktig hårddisk som du vill ansluta till ditt system. När du skruvar i den i din maskin och ansluter den får den ett blockenhetsnamn automatiskt, något i stil med */dev/hdb*. Sedan partitionerar du den med t.ex. **cfdisk /dev/hdb**, och får då flera partitioner. (Kanske vill du bara använda hela hårddisken som en stor partition, smaken är delad.) *Därefter* skapar du ett filsystem på disken med exempelvis **mkfs -t ext3 /dev/hdb1** (för ett ext3-system på första partitionen).[47]

mkfs kan även skapa filsystem *inuti filer*, som beskrivs t.ex. i avsnittet om CD-bränning på sidan 274.

Svenska tecken i filsystem

Ext3 och andra nya filsystem för hårddiskar är oftast standardmässigt inställda för att använda teckentabellen UTF-8 Unicode för att lagra filnamn. Den gamla standarden var oftast ISO 8859-1. Förändringen har införts eftersom den ger goda möjligheter till internationalisering (anpassning av text och liknande i användargränssnittet) och lokalisering (anpassning av t.ex. vikter, tidangivelser och datum) av systemet, bland annat för kinesiska och hindi.

UTF-8 är ett teckenkodningsystem[48] som gör det möjligt att pressa in unicodetecken, som normalt är sexton bitar långa, i åttabitarssekvenser, samtidigt som det är möjligt att använda de 128 första tecknen som om de fortfarande var ISO 8859-1. Detta är mycket bra för den anglosaxiska världen, eftersom deras språk kan skrivas med enbart de 128 första tecknen (som bland annat innehåller bokstäverna ”a” till och med ”z”, och därför normalt inte behöver ändra någonting alls) men lite jobbigare för oss andra som använder svenska tecken med prickar, ringar och accenter. Just *våra* specialtecken fanns i och för sig i ISO 8859-1, men i Unicode kodas de på ett annat sätt, eftersom de hade ett teckenvärde över 128.

[47]För att göra ett filsystem tvärs över en hel hårddisk, kör t.ex. **mkfs -t ext3 /dev/hdb** utan partitionsnummer efter enhetsnamnet.

[48]Teckenkodningssystem (teckentabeller), baseras på att varje tecken i alfabetet som skall kodas representeras av en siffra.

```
#!/bin/bash
IFS=$'\n'
FOO=`find . -maxdepth 1`

for OLD in ${FOO}; do
    NEW=`echo ${OLD} | iconv -f iso8859-1 -t utf8`
    if [ "$OLD" != "$NEW" ]; then
        echo "$OLD -> $NEW"
        echo "Byt namn (j/n)?"
        read IN
        if [ "$IN" = "j" ] || [ "$IN" = "J" ]; then
            mv ${OLD} ${NEW}
            echo "Bytte namn"
        fi
    fi
done;
```

Figur 5.9: Skript som (förhoppningsvis) kan hjälpa till att byta namn på filer från teckentabellen ISO 8859-1 till UTF-8.

En huvudsaklig anledning till att bytet gjordes var att ISO 8859-1 bara kunde lagra 256 olika tecken totalt. Enkel matematik ger vid handen att t.ex. kinesiska tecken, som är flera tusen till antalet, aldrig skulle kunna få plats. Därför har Unicode utarbetats, så att alla språk skall kunna samsas i en och samma teckentabell. Övergången skapar en del konflikter.

Vilket teckenkodningssystem som används i ett POSIX-system bestäms av miljövariabeln $LANG. Normalt skall denna vara satt till värdet **sv_SE.UTF-8** eller på äldre system **sv_SE**. Det senare kommer att resultera i att den gamla ISO 8859-1-teckenkodningen används. (kolla din inställning med **echo $LANG**, se vidare sidan 99.)

När du installerar ett helt färskt GNU/Linux-system vållar detta sällan några större problem. Problem uppstår däremot om du vill kunna hantera gamla filsystem med en teckenkodning som inte är UTF-8.

- Om du monterar ett UTF-8-filsystem på en system som använder ISO 8859-1 kommer det att se lustigt ut. Tecken som t.ex. Ã¶ istället för bokstaven ö är ganska vanligt.
- Om du monterar ett ISO 8859-1-filsystem på ett system som använder UTF-8 blir effekten att alla svenska tecken tycks falla bort, försvinna helt, eller dyker upp som frågetecken (t.ex. *fr?getecken*). Anledningen är att ensamma 8-bitarstecken inte är giltiga UTF-8-tecken.

Om du flyttar en partition permanent från ISO 8859-1 till UTF-8

måste du byta namn på *alla* filer och *alla* kataloger som innehåller svenska tecken. Skriptet i figur 5.9 kan kanske hjälpa till. Detta byter namn på filerna i den aktuella katalogen från namn enligt ISO 8859-1 till namn enligt UTF-8.

Om det rör sig om en enstaka användare, kan det räcka med att den användaren sätter sin $LANG-variabel till **sv_SE** istället för **sv_SE.UTF-8**. Detta kan t.ex. göras i användarens hemkatalog, i filen **.bash_profile** med raden **export LANG=sv_SE** (förutsatt att BASH är användarens standardskal!)

Om du behöver montera gamla VFAT- eller NTFS-partitioner (Microsoft Windows), eller gamla CD-ROM-skivor som inte använder UTF-8, kan detta lösas genom att ange monteringsalternativet **utf8** som t.ex. **mount -o utf8 /mnt/cdrom**. Detta kan också anges med parametrarna till en monteringspunkt i */etc/fstab*.

5.4.11 Hantering av moderkortet

Den del av din dators hårdvara som ligger allra närmast operativsystemets kärna är naturligtvis *moderkortet*. Direkt på moderkortet finns all hantering av hård- och mjukvarustyrda avbrott, direkt minnesaccess, det fysiska primärminnet i form av kapslar, och alla de bussar där grafikkort, nätverkskort och annan kringutrustning ansluts.

Viss kringutrustning kan sitta fastlödd direkt på moderkortet men eftersom denna oftast ändå rent konceptuellt hanteras av kärnan som om den vore en perifier enhet spelar det ingen roll. Hit hör bland annat integrerade grafik- och nätverkskort, USB-hubbar, skrivarport, serieport för modem o.s.v.

Viss hårdvara är däremot knuten mycket nära till moderkortet. Det gäller framför allt:

Strömsparfunktioner — dessa behöver kunna stänga ned stora delar av systemet för att det skall kunna *suspendera* (släcka aktivitet tillfälligt) eller *hibernera* (släcka aktivitet för en längre tid).

Frekvensskalning av datorns mikroprocessor. För att spara ström, vilket är speciellt viktigt på bärbara datorer, kan moderna moderkort steglöst ändra processorns klockhastighet beroende på hur stor last som för tillfälligt ligger på systemet. En hög frekvens på processorn leder nämligen till att den drar mer ström.

Sensorer — dessa kan känna av temperaturer i systemet och varvtalet på fläktar som är anslutna till moderkortet och stänga ned hela systemet om någon del överhettas eller någon fläkt stannar.

Knappar såsom ”strömbrytaren” på en bärbar dator som inte bryter någon ström utan skickar en signal till operativsystemet om att stänga ned datorn. (Dessa kallas ibland ”softkeys”, som i ”mjukvarustyrda knappar”.)

I äldre datorer finns ofta ett strömsparsystem kallar APM (Advanced Power Management) medan nyare datorer så gott som uteslutande använder ACPI (Advanced Configuration and Power Interface).[49] Operativsystemet är tvunget att interagera med ACPI: en dator som använder detta fungerar inte utan ett matchande operativsystem. APM-baserade datorer överlever dock ofta utan ett APM-medvetet operativsystem, eftersom hanteringen av strömsparfunktioner där istället sköts av datorns BIOS. ACPI-baserade system använder även de datorns BIOS för att genomföra olika åtgärder, men kontrollen över skeendet sköts av operativsystemet istället för direkt av BIOS.

ACPI dök först upp på bärbara datorer, och en markant egenhet med datorer som är utrustade med ACPI är att de verkligen stänger av datorn när du väljer ”stäng ner” från systemets menyer eller skriver POSIX-kommandot **shutdown -h now**. På äldre datorer måste du ofta trycka på någon mekanisk knapp för att verkligen stänga av strömmen till systemet så att fläktar o.s.v. slutar snurra. ”Avstängningsknappen” på en ACPI-utrustad dator skall bara skicka en signal till operativsystemet om att detta i sin tur skall stänga av systemet med hjälp av andra ACPI-anrop. Om det inte fungerar (om t.ex. systemet har ”hängt sig”) brukar det vanligen fungera att hålla knappen intryckt en längre stund för att verkligen *stänga av* strömmen.

Sensorer för temperaturer och fläkthastigheter skall normalt vara anslutna till APM eller ACPI men av historiska skäl är det ibland så att de måste hanteras direkt av operativsystemet istället, ofta via speciella drivrutiner som tillhandahållits av tillverkarna. Detta har skapat stora problem för såväl GNU/Linux som andra mer ovanliga operativsystem: tillverkarna väljer nämligen ofta att inte utveckla drivrutiner annat än för ett operativsystem.

I och med att ACPI blir allt mer av en branschstandard är dessa problem på väg att försvinna. ACPI-stödet i Linux är till stora delar skrivet av Intel, som också är ett av företagen som uppfunnit det, men det har inte blivit helt fungerande förrän i senare versioner av 2.6-versionen av Linuxkärnan.

[49] ACPI används som namnet antyder naturligtvis även för konfiguration. Den konfiguration som avses är inställningar av en del hårdvara i systemet vid uppstart. ACPI var ursprungligen tänkt som en delmängd av ett större koncept med namnet Operating System-directed configuration and Power Management (OSPM) men det senare har världen aldrig sett särskilt mycket av.

För temperatur- och fläktsensorer som inte använder ACPI eller APM finns separata drivrutiner i Linuxkärnan som hanterar många vanliga chip direkt. Projektnamnet för dessa är *lm_sensors*.[50] Det underliga namnet kommer sig av att de första sensorer som stöddes av projektet var av typen LM78 och LM75 från National Semiconductor. Det finns en uppsättning användaprogram knutna till lm_sensors, och dessa måste ofta installeras som ett separat programpaket för att du skall kunna ha nytta av sensorerna. I synnerhet är programmet **sensors-detect** användbart, eftersom detta automatiskt kan undersöka din hårdvara för att lokalisera sensorer av olika slag.

Sensorerna är oftast anslutna till moderkortet via en s.k. *SM-buss* (engelska *System Management Bus*, systemhanteringsbuss, ofta förkortat SMBus). I de fall lm_sensors behöver användas kan **sensors-detect** oftast hitta en lämplig drivrutin bland Linuxkärnans moduler för att komma åt SM-bussen, och i de fall där systemet är utformat för att använda sensorer via ACPI är bussen oftast ansluten direkt till den hårdvara som sköter ACPI, och du behöver därför inte bekymra dig om saken. Ibland är sensorerna tillgängliga på båda visen (dvs både genom ACPI och genom speciella SM-buss-drivrutiner) och då bör du välja att använda dem via ACPI.

Det räcker dock inte med att Linuxkärnan kan hitta sensorerna eller ACPI-hårdvaran för att ditt system skall använda det på ett bra vis. Kärnan vet inte mycket om hur den skall reagera på det sensorer och ACPI meddelar utan överlämnar åt resten av systemet att sköta detta på bästa vis. Alla distributioner levererar inte i dagsläget någon färdig, fungerande lösning för detta, men i t.ex. Debian används en uppsättning skript med namnet **laptop-mode** för detta, medan GNOME har utvecklat en egen strömhanteringskomponent på hög nivå med namnet *GNOME Power Manager*.

Sensorer kan både GNOME och KDE övervaka med indikeringar av olika temperaturer direkt på skrivbordet eller i någon verktygsrad för den som är intresserad. Dessa program brukar sällan kunna vidta några åtgärder om maskinen t.ex. blir överhettad så nyttan med dessa är snarast kosmetisk.

---

[50]Se `http://secure.netroedge.com/~lm78/`

KAPITEL 6

Fönstersystemet X

Officiell hemsida: `http://www.x.org/`

I maj år 1983 påbörjade Massachusetts Institute of Technology i Boston, USA projektet *Athena*, i syfte att integrera datorer med undervisningen vid denna tekniska högskola. Idag kan detta tyckas vara en självklarhet, men under det tidiga 1980-talet var datorer ännu inte något självklart inslag i ingenjörsutbildningen.

Som ett led i projektet skulle stora mängder datorer placeras ut på universitetet, alla elever skulle få egna datorkonton, och diverse olika grafiska datorprogram skulle göras tillgängliga. Som bas användes POSIX-liknande operativsystem.

När projektet avslutades år 1991 hade en mängd biprodukter skapats, den mest framgångsrika kom att bli det som idag är känt som *X Window System*, fönstersystemet *X*.[1] Detta system var influerat av ett tidigare fönstersystem kallat *W* skrivet av Paul Asente i Stanford och kallades därför *X* eftersom den bokstaven kommer efter W i alfabetet. Detta fönstersystem skapades genom ett samarbete mellan institutionen för datorvetenskap vid MIT där Jim Gettys bedrev arbetet med Athena, och företaget Digital Equipment Corporation. Projektledaren hette Robert Scheifler.

Den huvudsakliga anledningen till att *X* skapades var att MIT hade begränsad budget för projektet Athena och därför ville använda billiga

[1] Namnet på systemet är *X* kort och gott, *X Window System* är egentligen ett förtydligande.

datorer som anslöts till centrala servrar som skulle tillhandahålla själva beräkningskapaciteten. De billiga datorerna skulle bara användas som *grafiska terminaler*, inte som *datorer*, d.v.s. programmen skulle köras på en annan dator och de billiga datorerna skulle bara erbjuda en plats där grafiken kunde visas. Grafiken skulle transporteras från de centrala datorerna till terminalerna genom ett nätverk, på samma vis som en vanlig textterminal visar text från en dator på annat håll.[2]

Den första allmänt tillgängliga versionen av *X* var version 10, kallat *X10*, som snabbt följdes av den mer fullödiga version 11, *X11* år 1987. Licensvillkoren för *X*[3] var mycket förmånliga och ledde till att systemet mycket snabbt blev *de facto*-standard för fönstersystem i POSIX-miljöer och konkurrerade ut alla andra system.

I oktober år 1991 släpptes version fem av *X*, mest känd under förkortningen *X11R5*, såväl i form av en specifikation som i form av en generöst tilltagen exempelimplementation. För IBM PC-kompatibla datorer baserade på Intels 80386-processor medföljde i denna version en implementation för UNIX-liknande operativsystem skriven av tysken Thomas Roell. Denna implementation hade namnet *X386* och var vid detta tillfälle uppe i version 1.2.

Vid ungefär samma tidpunkt tog Roell all källkod till X386 med sig och flyttade till USA, där han snart började utveckla och sälja en proprietär version av programmet under namnet *Accelerated-X*, så ingen väntade sig att författaren i framtiden skulle bry sig om att undehålla X386. Eftersom X386 dessutom innehöll flera fel, var det svårt att ens använda programmet utan ändringar.

I april 1992 kom ett flertal utvecklare, som kommit i kontakt med varandra på Internet, överens om att fortsätta utveckla X386 med Roells kod som grund. Den första förbättrade versionen av X386 kallades *X386 1.2e* där e:et stod för ”enhanced”. Strax därefter byttes projektnamnet till *XFree86* i betydelsen *gratis*, mest på skoj, eftersom Roells version av X386 inte längre var gratis.[4] Bland dessa ursrpungliga utveckare av XFree86 fanns David Wexelblat, David Dawes Glenn Lai och Jim Tsillas.

Utvecklarna av XFree86 ville ursprungligen att deras *X*-server skulle distribueras tillsammans med det officiella *X*, men fick inte, eftersom de inte var något företag. De fick inte ens vara med i organisationen X Consortium eller The Open Group, och skapade då ett företag enbart för att kunna bli det. På medlemssidan för X.org låg XFree86 länge längst ned, trots att de oomtvistligen var den viktigaste medlemmen.

[2]Se avsnitt 2.3.11 på sidan 105 för mer detaljer om terminaler.
[3]Den s.k. MIT-licensen, se 3.4.3 på sidan 125.
[4]Om du på engelska uttalar X386 ”*X Three Eighty-Six*” respektive XFree86 ”*X Free Eighty-Six*” ser du vari ordvitsen i detta namn ligger.

Organisationen inom XFree86 har sedan dess varit flytande, och arbetstempot och koordinationen bland utvecklarna av varierande kvalitet. XFree86 har blivit något av GNU/Linux-världens sorgebarn. Under år 2003 började en spirande separatism prägla projektet, där några vill radikalt skriva om XFree86 och fann det svårt att göra detta inom ramen för projektet.

Huvudpunkten i kritiken som riktades mot XFree86 och *X* var bland annat att *X* blev övertungt på grund av all sin nätverkstransparens, d.v.s. att alla grafikanrop från olika program måste formuleras om av gränssnittet som om de skickades via nätverk. David Dawes föreslog därför att en direktrendringsmotor (som innebär att programmen har direkt tillgång till grafikkortet via ett enkelt gränssnitt) skulle utvecklas istället, och en enkel *X*-server byggas ovanpå denna, istället för att behålla all hantering av grafiken inuti *X*. Detta är i praktiken vad som åstadkommits i servern *Xgl*, som utvecklats till större delen mellan åren 2004 och 2006 av Novell och där David Reveman från Umeå Universitet haft en tongivande roll. Ett snarlikt projekt med namnet AIGLX har startats av Red Hat. Dessa projekt kan mycket väl vara framtiden för *X*, men är beroende av ett OpenGL-kapabelt grafikkort,[5] vilket gör att det i dagsläget inte kan köras på äldre datorer.[6] Xgl kom emellertid inte till inom ramen för XFree86, som vi snart skall se.

XFree86 är när detta skrivs uppe i version 4.5.0. Emellertid blev schismen mellan olika utvecklare år 2004 så stark att huvuddelen av utvecklarna lämnade XFree86 och återgick till organisationen X.org. Med sig tog de en uppdaterad variant av version 4.4.0rc2, eftersom licensen till XFree86 ändrats till en version som kräver namngivning av utvecklarna i samband med användandet. Denna licens är bl.a. inkompatibel med GNU GPL, och accepteras varken av GNU/Linux-distributörerna eller BSD-projekten. Licensförändringen var dock inte det enda skälet till att projektet splittrades; huvudskälet var som tidigare sagts att flera utvecklare upplevde att projektet avstannat. Snart efter det massiva avhoppet meddelade alla större distributioner samt BSD-projekten att de hädanefter skulle leverera sina system med X.org-servern istället för XFree86.

Stora delar av programbiblioteken som sköter specifika delar av funktionaliteten i XFree86 existerade sedan länge i egna, separata projekt.

---

[5]De OpenGL-baserade X-servrarna använder OpenGL mer eller mindre direkt. Anledningen är att 2D (en skrivbordsyta) egentligen bara är ett specialfall av 3D, och framtida grafikkort kan komma att enbart stödja 3D i en eller annan form.

[6]Såväl kommande Windows Vista som framtida MacOS X kommer att använda en direkt rendering av liknande slag, d.v.s. all grafik sköts av hårdvaran. Under GNU/Linux och MacOS sker detta med hjälp av OpenGL, medan Windows använder det Microsoft-specifika Direct3D.

Dit hör t.ex. FreeType och Fontconfig som ansvarar för större delen av typsnittshanteringen. Dessa projekt har i dagsläget börjat samlas kring webbportalen *freedesktop.org* för att med en helhetssyn som inkluderar såväl *X* som skrivbordsmiljöerna GNOME och KDE arbeta och sprida information om arbetet med de grafiska systemen i framför allt GNU/ Linux och BSD.

Stora delar av drivrutinerna för olika grafikkort som finns i *X* har skrivits av företaget Precision Insight som startats av Red Hat m.fl. för att förbättra grafikstödet i XFree86 och X.org-servern. Detta företag anlitas i sin tur av grafikkortstillverkare som vill ha stöd för sina grafikkort i *X*, däribland ATI[7], Intel, 3dfx och Matrox. Den huvudsakliga anledningen till att dessa företag stödjer XFree86 har varit att grafikkorten används mycket i professionell grafikproduktion i bl.a. Hollywood. De enda större grafikkortstillverkare som varit ovilliga att samarbeta direkt med X.org eller Precision Insight är nVidia och till viss del även ATI som istället för att på normalt vis inkorporera sina drivrutiner i X.org-servern har valt att leverera förkompilerade drivrutiner som inte är fri programvara och därför måste laddas ned och installeras separat. Detta skapar ofta problem eftersom binära drivrutiner måste anpassas till varje ny version av *X* och ofta även för den kompilator som använts för att kompilera *X* och det underliggande operativsystemet.[8]

Ägarskapet till den ursprungliga referensimplementationen av *X* har under åren vandrat från MIT till den självständiga organisationen *X Consortium* (1988–1996), till *The Open Group* (1996–1999), till *X.org* som numera har hand om specifikation och referensimplementation av *X*.

Den nuvarande versionen av referensimplementationen har revisionsbeteckningen *X11R7.0*. Denna namngivning skall utläsas så att protokollet som *X* använder har stannat vid version 11, och att enbart mindre revideringar har gjorts sedan dess. De sista siffrorna indikerar versionen av själva mjukvaran.

X är också unikt på det viset att samma program används tillsammans med GNU/Linux som med BSD-varianter, så att detta system är nödvändigt för båda dessa användargrupper. *X* är t.o.m. portat till Microsoft Windows, där det ingår som en del i det s.k. Cygwin-paketet.

---

[7] ATI är en akronym för Array Technologies International

[8] Det skall noteras att det förmodligen inte är något tjuvhållande på kod som är orsaken till nVidias och ATIs beteende: orsaken är att de två företagen konkurrerar oerhört hårt om varandras kunder och att de är ungefär lika stora. Ovilligheten att dela med sig av kod handlar om att ingendera företaget vill ge värsta konkurrenten insyn i sin källkodsbas, eftersom detta dels kan ge ledtrådar om "tänket" på företaget och därmed kan användas för att förutse framtida produktplaner m.m., dels att det ger en unik möjlighet att röna insikt i om det andra företaget har gjort patentintrång mot det egna företaget, eftersom många tekniker som används för just 3D-grafik är patenterade.

Figur 6.1: Logotypen som används av fönstersystemet *X*.

6.1 Arkitekturen i X

För att hålla isär begreppen tillämpas från och med nu följande konvention: när jag i följande text skriver *X* menar jag saker som hör till det generella systemet från X.org, medan jag väljer att skriva *X.org-servern* om det gäller detaljer som är specifika för X.orgs referensimplementation av *X*.

Fönstersystemet *X* bygger på följande grundläggande principer:

- **Nätverkstransparens** — datorprogram som är skrivna för att använda *X* kan alltid köra på en dator och visa grafik på en annan dator.

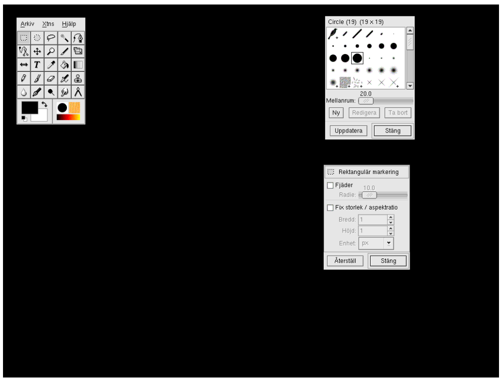
  I grunden består *X* av ett nätverksprotokoll som används av ett programmeringsgränssnitt[9] för programspråket C.

- **Klienten är en server** — det som normalt skulle kallas för "klient" ur ett användarperspektiv, kallas i X-världen för *server*, eftersom *X* ser världen ur programmens perspektiv istället för ur användarens perspektiv.

  En *server* är således en dator som tillhandahåller en bildskärm, ett tangentbord och en mus, medan en *klient* är ett program som ansluter sig till denna server.

  Klienten *kan* vara ett program på samma dator som kör X-servern, men behöver inte vara det. Även program som bara använder den

[9]Programmeringsgränssnitt heter på engelska *application programming interface* och förkortas därför ofta *API*



Figur 6.2: Den mest avskalade X-servern, den primitiva skärm som *X* erbjuder, har detta spartanska utseende. Här med programmet GIMP startat. Som synes finns varken fönsterkanter, knappar för att stänga fönster eller liknande. Markören (som inte syns här) har formen av ett *X*.

> skärm och det tangentbord som sitter på datorn där det körs, behandlar dessa som om de vore enheter enbart tillgängliga via nätverket och tar inga "genvägar".

X är *inte* ett grafiskt användargränssnitt. Grafiska användargränssnitt, som även kallas grafiska skal, eller på engelska *graphical user interfaces* (GUI:s), är system för att ge kommandon till datorn i samma anda som vid en kommandorad, med skillnaden att tangenttryckningarna har ersatts med grafiska byggelement och symboler som kan aktiveras med hjälp av en mus, på det vis som de flesta av oss är vana att använda datorer.

Vad *X* erbjuder är snarast en skärm, och möjligheten att rita saker på den. En avskalad X-server (som det program och den skärmbild som genereras av en X-server logiskt kallas) har utseendet som återfinns i figur 6.2. Program som skall startas i ett sådant avskalat system måste startas från en terminal, och finns ingen sådan att tillgå kan det upplevas som att datorn helt enkelt har låst sig: det finns en markör formad som ett *X* och en svart bakgrund, inget mer.

Eftersom detta knappast är vad användare förväntar sig av ett grafiskt system har flera alternativa *skrivbordsmiljöer* (engelska: *desktop environment*) utvecklats för att ge det utseende med ett skrivbord, ikoner, verktygsrader o.s.v. som vi är vana vid. Dessa kallades urspungligen *fönsterhanterare* och var då bara enklare system för att presentera menyer, hänga på knappar på fönster så att de kunde stängas och lister så att

de kunde flyttas, modifiera pekaren beroende på vad som finns under den o.s.v.

När fönsterhanterarna utökades med möjligheter att bygga och integrera applikationer (t.ex. med drag-och-släpp), hantera filer och prata med olika resurser i operativsystemet utan att använda POSIX-gränssnittet direkt, samt en uppsättning standardprogram (miniräknare, texteditor etc.) började de kallas skrivbordsmiljöer. Dessa kommer att avhandlas i avsnitt 6.6 på sidan 258 och framåt.

Vissa användare föredrar ännu de enklare fönsterhanterarna. TWM (Tiny Window Manager), FVWM2 (F[10] Virtual Window Manager), Window Maker, AfterStep, Enlightenment, GNUStep o.s.v. är namn på sådana enkla fönsterhanterare. Få vanliga användare använder numera något annat än den fönsterhanterare som kommer med skrivbordsmiljön.[11]

För en programmerare är gränssnittet och protokollet i *X* mycket primitivt: du kan be *X* att rita en linje, en kvadratisk yta, en cirkel o.s.v., men de avancerade grafiska byggelement, s.k. *widgets* som används av moderna grafiska program finns inte i *X*. Dessa har därför utvecklats separat i olika programpaket, som i sin tur använder *X*. Motif, GTK+ och Qt är tre sådana paket, som kommer att beskrivas senare i detta kapitel.

Det mest grundläggande objektet i *X* är *rotfönstret* som ligger bakom alla andra fönster som en svart eller miniatyrschackrutig platta.[12] Detta har samma storlek som den upplösning din bildskärm är satt till, och kan inte stängas. Den svarta bakgrunden i figur 6.2 är rotfönstret. Ovanpå detta kan program öppna och stänga egna fönster. Allt du ser i *X*, utom markören, är uppbyggt av sådana fönster.

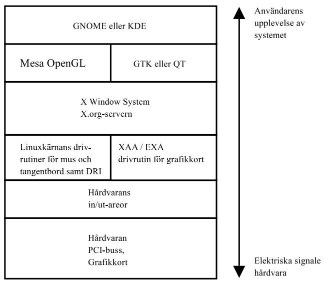
Specifikt för X.org-servern, och den kombination av program som används i GNU/Linux på en standardiserad IBM PC-kompatibel dator, har hierarkin av delsystem som bygger upp det grafiska användargränssnittet ett utseende i stil med det som återfinns i figur 6.3.

Små inbyggda system använder inte *X*, istället använder sådana system som behöver grafik (många inbyggda system nöjer sig med mycket enkla textterminaler) en s.k. *framebuffer*, vilket i korthet innebär att

[10]Upphovsmannen till FVWM, Rob Nation, har glömt vad *F* i förkortningen stod för.

[11]Undantagen är a) entusiaster som älskar en viss fönsterhanterare och menar att skrivbordsmiljön är meningslös eller överflödig, b) de som använder gamla, långsamma datorer, och c) de som tvärs emot alla rekommendationer byter ut den fönsterhanterare som finns inbyggd i skrivbordsmiljön mot en annan.

[12]Det är även möjligt att köra *X* utan rotfönster, detta är vad exempelvis Cygwin X och den *X*-server som körs på MacOS gör. *X*-fönstren blir då istället en del av det underliggande fönstersystemet (som då inte är *X* utan t.ex. Windows eller MacOS egna fönsterhanterare).



Figur 6.3: Hierarkin som döljer sig bakom användargränssnittet i GNU/Linux. Längst ner den faktiska hårdvaran och dess grafikkort, längst upp det som en användare till sist upplever. I framtiden kommer säkerligen renderingsmotorer som *Cairo* och *Arthur* samt bottenlager baserade endast på OpenGL att bli vanligare.

Linuxkärnan helt tar över ansvaret för grafiken.[13] Det finns även en nedbantad version av *X* med namnet *tinyX* som används i en del inbyggda system med högst rudimentära grafikbehov. Inbyggda system som inte har strikta minneskrav brukar i allmänhet använda någon variant av den vanliga *X*-servern från X.org.

6.2 Konfiguration

En mycket speciell egenhet med X.org-servern på GNU/Linux, och som framgår av figuren, är att Linuxkärnan normalt *inte* hanterar grafikkortet, i varje fall inte när det gäller att köra *X*. Istället har *X* egna drivrutiner till de flesta grafikkort, och använder dessa direkt. Något stöd för grafikkortet i kärnan behövs inte, och kärnan är bara utformad för att ge grundläggande POSIX-funktionalitet. *X* kommer däremot att använda mus och tangentbord via kärnan i någon mån.[14]

När basen i ett GNU/Linux-system är installerat måste därför X Window System konfigureras separat, även om resten av operativsystemet redan är på plats.[15]

[13]Vissa Linuxprogram med höga prestandakrav kan också använda framebuffern, som ursprungligen skapades då Linuxkärnan portades till Macintosh-datorer. Eftersom Mac:arna saknade textläge var utvecklarna tvungna att emulera text på en ren grafikyta, och resultatet blev framebuffern. Denna inkorporerades senare även i Intel x86-delen av Linux.

[14]Även om det t.ex. använder egna tangentbordskartor för att koppla samman en tangent med ett visst tecken.

[15]Före version 7.0 av *X* använde programmen använder inte ens GNU:s eller andra

Som vanligt hjälper distributionernas konfigureringsprogram i viss mån till med konfigurationen, och detta är en av deras stora fördelar. Såväl Red Hat:s *Anaconda* som SuSE:s *YaST* är mycket bra på detta, medan andra distributioner som Debian GNU/Linux eller Gentoo Linux ger varierande grad av hjälp som i princip kräver detaljerad kunskap om det mesta som avhandlas i detta kapitel och mer därtill.

Innan du börjar testa olika konfigurationer bör du läsa lite om olika program som ingår i *X*-sviten, i synnerhet om **Xorg** självt på sidan 246 och absolut om XDM på sidan 248. Även programmet **startx** som används för att starta upp X.org-servern om inte systemet gör det automatiskt, kan vara bra att känna till.

Med X.org-servern följer inte mindre än tre olika konfigurationsprogram, inget av dem är perfekt. Om din distribution inte genererat, eller misslyckats med att generera en konfigurationsfil, är detta din enda väg ut. Du bör använda ett av dessa program för att generera en grundläggande konfiguration som du sedan editerar för hand med en texteditor:

Xorg -configure kan, och *skall* normalt köras för att producera en grundläggande konfigurationsfil. Kommandot kommer att försöka autodetektera vilket grafikkort och vilken monitor som är ansluten till datorn, och utifrån detta genererar det en konfigurationsfil.

En stor fördel med detta kommando är att det genererar en bortkommenterad lista med olika flaggor till grafikkortet, något som annars kan vara stört omöjligt att finna dokumentation på. Dessa kommer standardmässigt att vara bortkommenterade och odokumenterade, men du kommer i alla fall att veta vilka flaggor som *finns*.

xorgconfigure (ursprungligen XFree86 **xf86configure**) är numera oftast borttaget i distributionerna, men var ett interaktivt program för att generera en konfigurationsfil. Tidigare var detta det enda sättet att generera en konfigurationsfil.

xorgcfg (tidgare XFree86 **xf86cfg**) är en nyare variant av samma program, något mera användarvänligt. Det är numera ofta borttaget ur distributionerna det också.

Om filen som genereras av kommandot **Xorg -configure** eller med något av de andra verktygen fungerar klanderfritt så är det bara att gratulera. Detta tillhör faktiskt ovanligheterna. Ibland fungerar det, men

POSIX-programs standarder för kompilering och installation, och levde därför oftast sitt eget liv i katalogen */usr/X11R6*, t.ex. installerades de körbara *X*-programmen i */usr/X11R6/bin* och de dynamiska länkbiblioteken i */usr/X11R6/lib*. Numera är de snyggt inlemmade i samma filträd som alla andra program.

upplösning och färgläge på *X* är inte vad de borde vara. Då återstår bara manuell konfiguration

Konfigurationen av X.org-servern utan verktyg kan inte påstås vara användarvänlig. Det är tvärt om något som av många användare, såväl noviser som avancerade användare, uppfattar som den mest mystiska och svårhanterliga delen av ett GNU/Linux-system.

Konfigurationen brukar i allmänhet finnas i */etc/X11* och den huvudsakliga konfigurationsfilen heter oftast */etc/X11/xorg.conf*. Konfigurationsfilen är indelad i *sektioner* som var och en har ett namn, och konfigurerar en viss aspekt av *X*. Varje sektion kan ha en *identifier* som ger den sektionen ett unikt namn. Denna *identifier* används sedan av andra sektioner för att skapa en referens till inställningar som gjorts i en annan sektion.

I varje sektion finns *variabler* som kan konfigureras, och som kan vara av fyra olika slag:

boolesk variabel — egentligen värdena *sant* eller *falskt*, men i *X*-världen kan *falskt* anges med något av orden *"False"*, *"Off"*, *"No"* eller bara *"0"* medan *sant* skrivs som *"True"*, *"On"*, *"Yes"* eller *"1"*. Booleska värden skall alltid anges med citationstecken kring själva värdet.

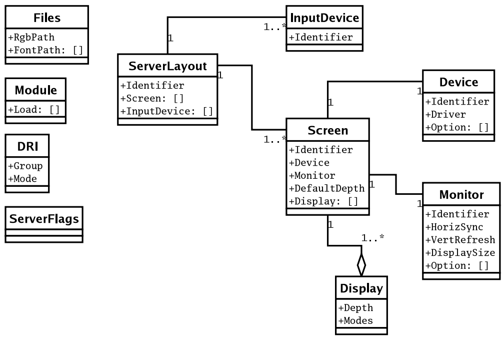
heltal — ett vanligt heltal, t.ex *50* Dessa skall anges *utan* citationstecken kring själva värdet, och kan även anges hexadecimalt genom att skriva t.ex. *0x2e*.

flyttal — ett numeriskt värde angivet med en punkt för att avskilja heltals- och decimaldel. Du bör då använda t.ex. *50.0* för att beteckna talet 50, inte bara *50*. Dessa anges *utan* citationstecken kring själva värdet.

sträng — är bara en följd av tecken. En sträng står *alltid* inom citationstecken.

Sektionernas inbördes ordning är inte viktig. De olika sektionerna kan ligga huller om buller, det viktiga är att de *finns* och att de kan *referera varandra* genom sina *identifier*-namn. Varje sektion skall uppfattas som ett *objekt* i samma anda som gäller inom objektorienterad programmering, och de inbördes objektrelationerna illustreras bäst av ett UML-diagram, se figur 6.4.

Option-variabler är speciella: dessa definierar dels ett namn i form av en sträng inom citationstecken, dels ett värde som skall sammankopplas med denna sträng (nyckel → värde). Om du t.ex. skriver: **Option "Foo" "Bar"** kommer nyckeln **"Foo"** att få värdet **"Bar"**. Själva *värdet* på *Option*-variabler kan ha alla typer som anges ovan, och i det



Figur 6.4: UML-diagram för objektstrukturen i konfigurationsfilen för X.org-servern. En *ServerLayout*-sektion behöver en eller flera *InputDevice*-sektioner och en eller flera *Screen*-sektioner. Varje *Screen*-sektion har en tillhörande *Device*-sektion och *Monitor*-sektion samt minst en *Display*-sektion. De övriga sektionerna svävar för sig själva och är inte beroende av andra sektioner.

fall det är fråga om en boolesk variabel kan denna anges utan värde, t.ex. **Option "Foo"**, och kommer då att tolkas som om du skrivit **Option "Foo" "On"**. Omvänt kan **Option "NoFoo"** användas ekvivalent mot **Option "Foo" "Off"**.

En kommentar kan infogas i konfigurationsfilen om den föregås av tecknet #. Detta används mycket för att kommentera bort saker, t.ex. **# Option "Foo" "Off"**.

Denna bit av konfigurationen brukar i allmänhet vara den första tröskeln. Därefter måste varje sektion konfigureras, och om det blir det minsta fel i någon sektion kommer inte *X* att fungera. Därför fordras en djup förståelse för var och en av sektionerna:

Files anger vart olika filer som används av X.org-servern finns. Följande variabler finns i denna sektion:

RgbPath (sträng) anger vart RGB-databasen som X.org-servern använder finns. Normalt är detta förinställt och skall inte ändras. Själva filen som anges innehåller ett antal namn på färger med namn som *LemonChiffon*, *LightSlateGray* o.s.v.[16] Dessa namn används av programmerare när de vill referera till vissa färger som definierats i *X*-specifikationen.

ModulePath (sträng) behöver sällan anges, men anger då den är

[16]Se filen */usr/share/X11/rgb.txt* om du vill se vilka färger som finns.

definierad sökvägen till de moduler som angetts i *Module*-sektionen, drivrutiner för grafikkort med mera.

FontPath (sträng) anger vart olika *fonter* (filer innehållande teckenuppsättningar[17]) för *X* finns lagrade. Det är även möjligt att ange en speciell server på nätverket som tillhandahåller fonterna, detta är speciellt bra vid central administration av tillgängliga fonter i ett större system, eller för disklösa klienter, s.k. X-terminaler,[18] som inte har plats för några stora fontmängder.

En plats i filsystemet anges med fullständig sökväg, t.ex. *"/usr/share/X11/fonts"*, och en server anges med protokoll och servernamn, t.ex. *"tcp/foo.bar.org"*. Eventuellt kan en port anges för fontservern, men i allmänhet används port 7100 för *X*-fonter från en server. Flera distributioner startar en fontserver på samma dator som kör X.org-servern, eftersom detta snabbar upp hanteringen av fonter. Då förekommer ofta värdet *"unix/:7100"*, vilket anger att en standard UNIX-sockel skall användas för att ansluta till port 7100 på samma dator.

(Se vidare sidan 253 om namn på fonter i *X*.)

Module — den här sektionen anger vilka moduler ("plug-ins") som skall laddas in i X.org-servern. Vissa delar av programmet kanske inte fungerar med den hårdvara som finns i din dator, och om du inte använder vissa delar av funktionaliteten i programmet, finns det heller ingen anledning att ladda in dem i minnet där de bara tar plats.

Load (sträng) anger att en viss modul skall laddas. Platsen som de laddas från kan anges i *ModulePath* i *Files*-sektionen, men är oftast fördefinierad till */usr/lib/xorg/modules/extensions* & */usr/lib/xorg/modules/fonts*, där en del av modulerna är lagrade i form av dynamiska länkbibliotek med namn som *libfoo.so*, och skall då laddas med variabeln **Load "foo"**.

Typiska moduler som brukar laddas är *dbe*, *dri* (direct rendering infrastructure, se nästa sektion) *extmod* (utökningar), *GLcore* (OpenGL), *glx* (OpenGL), *record* o.s.v.

Notera att för att DRI skall fungera måste såväl *glx* som *dri* vara inladdade i X.org-servern.

Det är ingen vidare ordning på moduler som används av X.org-servern, och därför är det normalt bäst att inskränka sig till att

[17] Teckenuppsättningar kallas även fonter, typsnitt, teckensnitt etc.
[18] se vidare sidan 256

kommentera bort eller avkommentera moduler som redan är uppradade i *Module*-sektionen.

DRI — denna sektion konfigurerar DRI, *direct rendering infrastructure*. En del av X.org-servern som ursprungligen skrevs av Precision Insight för att förbättra prestanda med OpenGL-grafik.[19] Denna del av X.org-servern är tätt integrerad med program från programbiblioteket Mesa (ursprungligen skrivet av Brian Paul), som implementerar själva OpenGL-gränssnittet.

DRI är tänkt att även användas för annan grafikintensiv hantering på samma vis som t.ex. DirectX i Microsoft Windows.

Om grafikkortet i datorn stödjer OpenGL kommer detta med en del tricksande att användas, men i annat fall kommer Mesa:s (långsammare) mjukvaruemulering av OpenGL att användas. Mesa distribueras tillsammans med X.org-servern och kommer därför att erbjuda OpenGL på alla datorer där det installeras, dock med varierande hastighet beroende på grafikkort.

När DRI används skapas en blockenhetsfil för varje grafikkort i datorn genom att X.org-servern kommunicerar direkt med Linuxkärnan. Blockenheterna kommer att finnas i */dev/dri/cardN* där *N* är ett nummer från 0 och uppåt, som numrerar de olika grafikkorten i datorn. Normalt finns bara */dev/dri/card0*. Du kommer också att se katalogen */proc/dri* i */proc*-filsystemet.

DRI kan konfigureras med följande variabler:

Group (heltal) anger en viss grupp som skall ha ägarskapet till blockenhetsfilerna */dev/dri/cardN*. Gruppen anges med en siffra som motsvarar gruppens ID-nummer i */etc/groups*. Normalt sätts detta till 0, vilket är gruppen för root-användaren.

Mode (4-siffrors heltal) anger filbehörigheter för blockenhetsfilerna */dev/dri/cardN*. Normalt 0666, d.v.s. läs- och skrivbart för alla användare.

Om du av någon bizzarr anledning tror att en annan användare skulle kunna få för sig att använda DRI-gränssnittet i din dator mot dig, kan du ange att bara en viss grupp skall ha tillgång till DRI-blockenhetsfilen, lägga till dig själv i denna grupp, och sätta filrättigheten till 0660.

[19] OpenGL är ett standardiserat programmeringsgränssnitt (API) för att programmera grafik, främst tredimensionell grafik, men även tvådimensionell grafik (som används en del i skrivbordsytor med stöd för tvådimensionell vektorgrafik) stödjs av OpenGL.

ServerLayout Denna sektion är normalt okomplicerad, men blir lite mer komplex om du använder s.k. *multi-head*-display.

Eftersom *X* bara presenterar en yta där program kan arbeta gör det inget om ytan sträcker sig över flera olika skärmar, eller flera olika grafikkort, så länge ytan definieras på ett vis som gör att *X* kan presentera den för program som behöver utrymme på skärmen. Det blir allt vanligare att professionella datoranvändare har mer än en skärm att arbeta med, och i *X* är detta inga problem att hantera. Exempelvis kan två grafikkort med 1024x768 upplösning[20] kombineras så att en total arbetsyta om 2048x768 skapas. Huvudsaken är att dessa passar samman.

ServerLayot är som du kan se i figur 6.4 roten till det träd av beroende objekt som grenas ut i *InputDevice*, *Screen* o.s.v.

Identifier (sträng) innehåller ett namn på din konfiguration. Eftersom det bara ska finnas en *ServerLayout* så har detta värde ingen betydelse, välj ett namn du tycker passar, t.ex. *"Foo"*.

Screen (sträng) anger en identifierare för vilken skärm som skall användas. Det är möjligt att använda flera skärmar samtidigt och kombinera dem till en yta. I sin enklaste form innehåller denna dock bara en rad i stil med: **Screen "foo"**.

Om du har flera *Screen*-rader måste de relateras till varandra på något vis och det hela blir genast mer komplext. För mer komplicerade exempel hävisas till manualsidan för själva filen xorg.conf[21] eller dokumentationen som följer med avancerade grafikkort.

En vanlig dator med *en* skärm skall endast ha *en* Screen-rad.

InputDevice (sträng) (sträng) anger olika inorgan som X.org-servern skall använda sig av. Normalt finns åtminstone två: en rad för musen och en för tangentbordet. Själva inorganen *i sig* definieras i sin tur i motsvarande *InputDevice*-sektioner. Vissa bärbara datorer har en extra musport för löstagbar mus och har därför ytterligare en rad, så att både den inbyggda musen eller styrplattan och den externa musen kan användas samtidigt.

Mus och tangentbord anges med sina respektive identifierare, t.ex. **InputDevice "minmus" "CorePointer"** och **InputDevice "mittkeyboard" "CoreKeyboard"** Notera att den här namngivningen på identifierare är helt godtycklig, bara de

[20] Upplösningen för grafikkort anges i antal bildelement (pixlar) i x- respektive y-led.

[21] Skriv bara **man xorg.conf**.

passar samman är det inga problem att kalla de olika inorganen för precis vad som helst. (Undvik dock svenska tecken och mellanslag för säkerhets skull, det funkar kanske, men alla är ändå försiktiga med att utmana ödet.)

Den andra parametern till en *InputDevice* med värdena **"CorePointer"** respektive **"CoreKeyboard"** skall bara anges för *exakt ett* inorgan. Markeringen talar om att detta är den *primära* musen och det *primära* tangentbordet. Vissa datorer har flera musar och kan därför fordra fler *InputDevice*-sektioner. Dessa "extramusar" (eller "extratangentbord" om det nu finns sådana) skall ha värdet **"AlwaysCore"** efter sin identifierare. Detta medför att *båda* musarna kan användas samtidigt. En del bärbara datorer har denna konfiguration.

InputDevice är sektioner som vardera definierar *ett* inorgan till din dator, t.ex. en mus, ett tangentbord, en touchpad o.s.v.

Eftersom inorganen normalt sköts av Linuxkärnan, skall du här ange vart dessa är anslutna i operativsystemet, oftast i form av teckenenhetsfiler någonstans i */dev*-delen av filhierarkin. Du måste också ange exakt vilken typ av tangentbord, mus o.s.v. det är fråga om, för även om din Linuxkärna och skalet är inställt för svenska tangentbord och en viss typ av mus, hanterar X.org-servern detta helt fristående och vet inget om hur Linuxkärnan eller skalet sköter saken. Det är alltså nödvändigt att upprepa sig.

Eftersom de flesta användare tillbringar mer tid i *X* än i t.ex. Linuxkonsollen är dessa inställningar kanske ännu viktigare än inställningarna i kärnan och skalet.

Identifier (sträng) är ett namn som skall anges i *ServerLayot*-sektionen för att "koppla in" inenheten.

Driver (sträng) talar om vilken drivrutin som skall användas för detta inorgan. Olika former av inorgan har olika drivrutiner, de vanligaste är:

Drivrutin	**Hårdvara**
"keyboard"	Tangentbord
"mouse"	Mus

Det finns andra, men de är inte speciellt vanliga. De har namn i stil med filerna som finns i */usr/lib/xorg/modules/input*. T.ex. kan en joystick användas av den som så önskar. En touchpad eller miniatyrstyrspak av det slag som finns på bärbara datorer räknas för det mesta som en mus.

```
Section "InputDevice"
        Identifier  "Foo"
        Driver      "kbd"
        Option      "XkbModel" "pc105"
        Option      "XkbLayout" "se"
EndSection
```

Figur 6.5: Definition av ett svenskt tangentbord. Önskas Svorak-layout kan värdet **"se"** ändras till **"dvorak(se)"**.

För varje typ av drivrutin måste sedan ett antal *Option*-variabler sättas, för att X.org-servern skall kunna veta hur många tangenter ett visst tangentbord har, om musen har ett extra skrollhjul, o.s.v. Dessa är mycket viktiga och blir tyvärr ofta fel.

Ett svenskt tangentbord har standardutseendet som framgår av figur 6.5, och olika tillämpbara *Option*-variabler för tangentbordet återfinns i figur 6.6.

En mus är lite knepigare att definiera, speciellt om en udda variant används. För en vanlig Microsoft-tvåknapparsmus som är inkopplad i PS/2-porten[22] bör dock sektionen se ut något i stil med figur 6.7. De olika *Option*-parameterar som kan användas för en mus finns uppradade i figur 6.8.

Screen är sektionen som konfigurerar den *skärm*, eller om det rör sig om flera sektioner, de *skärmar* som används av *ServerLayout*-sektionen. Det måste *alltid* finnas *minst en Screen*-sektion. Följande konfigurationsrader skall konfigureras:

Identifier (sträng) anger det namn som skall användas för referera till denna skärm från *ServerLayout*-sektionen, t.ex. **"Foo"**.

Device (sträng) anger namnet i identifieraren på den *enhet* som skall hantera denna skärm. Enheten är detsamma som ett grafikkort, och definieras i en separat sektion.

Monitor (sträng) anger namnet i identifieraren på den *bildskärm* som skall användas tillsammans med den enhet som också angivits, t.ex. **"Bar"** Innebörden av denna och föregående variabel blir sammantaget *jag har grafikkortet* **"Foo"** *anslutet med en kabel till skärmen* **"Bar"**.

[22] PS/2-porten (som heter så eftersom IBM var de första som använde den i sina PS/2 datorer) är samma typ av port med en miniatyriserad DIN-kontakt som används av en modernare dators tangentbord.

Option	Värden
"XkbRules"	**"xorg"** är det värde som denna variabel i stort sett alltid har, om den ens förekommer. Om du installerar på en Sun SPARC kommer den att ha värdet "sun" men det tillhör väl ovanligheterna. Om du skulle används XFree86 står här istället **"xfree86"**.
"XkbModel"	**"pc105"** är det normala, svenska tangentbordet. Om du räknar antalet knappar skall de vara 105 stycken, och inkluderar då tre knappar med varsin Microsoft Windows-logotype och en "menytangent". Om dessa tre saknas har du istället ett äldre tangentbord av typen **"pc102"**.
"XkbLayout"	Här används en geografisk landskod enligt ISO 3166 som gäller för att tala om vilka bokstäver som finns på de olika knapparna. Om du talar svenska är det vanligtvis **"se"** som gäller. Om du vill använda svensk Dvorak-layout (ibland kallad *Svorak*) är det **"dvorak(se)"** som skall stå här. Se tabellen på sidan 251 för fler intressanta alternativ. (Det är lämpligt att ditt val av tangentbord matchar ditt val av språk för systemet.)
"XkbVariant"	**"nodeadkeys"** är det enda värde denna variabel kan sättas till, vilket deaktiverar möjligheten att lägga accenter med hjälp av de tangenter som normalt inte producerar några tecken, t.ex. att komponera den franska bokstaven **Ô** genom att först trycka på ˆ och sedan på **O**. Det som kallas för *döda tangenter* är alltså accentueringstangenterna uppe till höger på tangentbordet.

Figur 6.6: Olika *Option*-variabler som kan användas i en *InputDevice*-sektion med *Driver*-variabeln satt till *"keyboard"*.

```
Section "InputDevice"
        Identifier  "Foo"
        Driver      "mouse"
        Option      "Protocol" "PS/2"
        # Byt ovanstående till "IMPS/2" för skrollhjulsmus
        Option      "Device" "/dev/input/mice"
        # Byt ovanstående till "/dev/psaux" under kernel 2.4
        Option      "Emulate3Buttons" "yes"
EndSection
```

Figur 6.7: Definition av ett inorgan i form av en vanlig PS/2- eller USB-mus.

Option	Värden
"Protocol"	Har för det mesta värdet **"PS/2"** eller **"IMPS/2"**. Det första anger att det är en vanlig mus, det andra att det är en mus med ett skrollhjul mellan de två tangenterna. Om du har en mus av det senare slaget *skall* du också använda variabeln **"ZAxisMapping"** nedan.
"Device"	anger vilken teckenenhetsfil i operativsystemet som erbjuder kommunikation med musen. **"/dev/input/mice"** är det generiska namnet för alla systemets musar, men **"/dev/psaux"** brukade anges för PS/2-musar under version 2.4 av Linuxkärnan.
"Emulate3Buttons"	anger om en tvåknappars mus skall kunna användas som en treknappars, genom att den tredje knappen (mittenknappen) representeras av ett tryck på båda tangenterna samtidigt. Detta kan vara användbart, eftersom många *X*-program använder mittenknappen flitigt för t.ex. kopiering.
"ZAxisMapping"	skall bara användas om du har en mus med skrollhjul, d.v.s. om **"Protocol"** också är satt till **"IMPS/2"**. Värdet **"4 5"** används normalt. Detta anger att knapp 4 och 5 på musen skall användas för att kontrollera skrollhjulet. Namnet kommer från att skrollhjulet anses utgöra en "tredje dimension" (Z-axeln) och knapp 4 och 5 är virtuella knappar som det genereras ett antal tryckningar på när skrollhjulet rullas upp eller ned.

Figur 6.8: Olika *Option*-variabler som kan användas i en *InputDevice*-sektion med *Driver*-variabeln satt till *"mouse"*.

Det kan tyckas onödigt att ange vilken enhet en skärm är ansluten till, men detta har att göra med att X.org-servern genom att jämföra prestanda hos enheten med bildskärmen kan avgöra vilka grafiklägen denna kombination klarar av att hantera.

DefaultDepth (heltal) anger standardfärgdjupet för denna kombination av enhet och bildskärm. För alla moderna komponenter har detta värdet **24** eller rent av **32**, och siffrorna anger hur många binära bitar som används för att representera ett färgvärde i en bildpunkt på bildytan. 24 bitars grafik innebär att $2^{24} = 16777216$ olika färgvärden[23] kommer att erhållas. Detta värde måste matcha *Depth*-variabeln i någon av *Display*-undersektionerna (se nedan), och det måste naturligtvis också stödjas av enheten (grafikkortet) som används.

Värdet *skall* vara en jämn binär multipel, eller jämt delbart med 8, d.v.s. i praktiken 1, 4, 8, 16, 24 eller 32. En del halvgamla grafikkort blir ibland snabbare och du sätter ned antalet bitar från 24 till 16, och om du inte jobbar med digitalt foto är skillnaden marginell.

Äldre grafikkort kunde ha värdet 8 här, eftersom de bara kunde hantera $2^8 = 256$ olika färger, och om du använder riktigt gamla *X*-program kan du få behov av stöd för s.k. *PseudoColor*, och detta kan erhållas genom att sätta *DefaultDepth* till 8, samt lägga till en motsvarande *Screen*-sektion. (Se vidare information under kommandot **xdpyinfo** nedan.)

Utöver detta finns alltid en eller flera undersektioner till *Screen*, med namnet *Display*. En *Screen*-sektion har det typiska utseendet i figur 6.9. Undersektionen *Display* innehåller alltid:

Depth (heltal) ett färgdjup, som för *DefaultDepth* ovan, 2, 4, 8, 16, 24 eller 32 bitar. Minst *en* av *Display*-undersektionerna måste ha ett färgdjup som matchar *DefaultDepth* på ovanliggande nivå.

Modes (strängar) följs av ett antal strängar som innehåller tillgängliga upplösningar för den färgmod som anges för denna *Display*, angivet i $X \times Y$-format.

Eftersom ett större antal färger kan medföra att det krävs mycket minne för att lagra färgerna, och att antalet bildelement, och därmed upplösningen blir lidande. Därför kan det

[23]Ibland slarvigt kallat ”sexton miljoner färger” trots att det är närmare sjutton.

```
Section "Screen"
        Identifier "Fnord"
        Device     "Foo"
        Monitor    "Bar"
        DefaultDepth  24
        SubSection "Display"
                Depth     24
                Modes    "1024x768" "800x600" "640x480"
        EndSubSection
EndSection
```

Figur 6.9: Definition av ett ganska vanlig skärm i *X*, med enbart 24 bitars färgdjup och en maxupplösning på 1024 × 768 bildelement.

t.ex. vara lägre högstaupplösning för 24-bitarsfärg än för 16-bitarsfärg, och därför får du möjlighet att på detta vis välja vad du vill ha.

Upplösningarna anges från vänster till höger, den som ligger längst till vänster (i exemplet ovan **"1024x768"** kommer att användas som standard. Skall du bara använda din skärm i en upplösning (t.ex. högsta upplösningen) så räcker det med ett enda värde, så länge det fungerar.

De tre upplösningarna som angetts i exemplet är något av standard, men du bör i din bildskärms manual kunna hitta vilka maxupplösningar den klarar av. Vissa skärmar hanterar upp till **"1280x960"** bildelement, och vissa **"1152x864"**. Om du är osäker kan du skriva in flera olika med den högsta först, för om du konfigurerat din enhet (grafikkort) och bildskärm korrekt, kommer X.org-servern att räkna ut ifall en upplösning är för hög för utrustningen och automatiskt att prova med näst högsta och så vidare nedåt, tills det hittar en upplösning som fungerar.

I tidigare versioner av XFree86 användes inte detta system med *Display*-sektioner, utan ett mycket mer komplicerat system med s.k. *modelines* i sektionen *Monitor* användes. Detta var betydligt mer komplicerat och som tur är slipper du i regel att använda detta. (*Modelines*-formatet kommer inte att redovisas här.)

Device anger konfigurering för ett grafikkort som används i din dator. X.org-servern kan använda flera grafikkort, men normalt har du bara ett.

Identifier (sträng) anger det namn som *Screen*-sektionen skall an-

vända för att referera till detta grafikkort.

Driver (sträng) är namnet på den drivrutin som skall användas för att kontrollera det här kortet.

De drivrutiner som följer med X.org-servern finns lagrade i katalogen */usr/lib/xorg/modules/drivers* och har namn i stil med *foo_drv.o*. Detta innebär att om ditt grafikkort använder drivrutinen med namnet *foo_drv.so* så är det **"foo"** som skall stå i *Driver*-variabeln. Vanliga rutiner är de som finns i figur 6.10.

Anledningen till att *kretstillverkare* (engelska: chipset manufacturer) är angiven, är att mängden världens samlade grafikkort är mycket stort, helt enkelt därför att olika tillverkare köpt in IC-kretsar och tillverkat kort som de sålt under eget namn. Uppsättningen med olika kretsar som kan användas för att tillverka sådana kort är däremot mycket begränsad, och korten fungerar ofta likadant, trots vitt skilda namn.

Om du inte har en aning om vem som tillverkat din kretsuppsättning går det alltid bra att skruva ur grafikkortet ur datorn och titta på det, eller om det sitter på datorns moderkort, inspektera moderkortet i närheten av videokontakten. Sökningar på Internet kan också hjälpa till. Det är inte säkert att Linux stödjer alla grafikchips som existerar, men en förkrossande majoritet kan användas, och fungerar inget annat så fungerar ofta VESA-drivrutinen, eller i värsta fall VGA-rutinen.

VendorName (sträng) skall innehålla namnet på grafikkortstillverkaren, t.ex. **"Foobar International"**.

BoardName (sträng) skall innehålla namnet på just detta grafikkort, t.ex. **"Foobar FunCard"**. Varken detta eller föregående är tekniska krav, utan mer för ditt eget vidkommande, samt för att felmeddelanden o.dyl. skall vara begripliga.

VideoRam (heltal) — för de flesta grafikkort krävs inte denna variabel, den används bara i drivrutiner till grafikkort som inte själva kan rapportera hur mycket minne som sitter på kortet. Om den finns med har den värden som anges i kilobyte: 1024 = 1MB minne, 2048 = 2MB minne o.s.v.

En fullständig *Device*-sektion kan se ut som i figur 6.11.

De flesta grafikkort tillåter en hel drös *Option*-variabler för att slå på eller av specialfunktioner i kortet. Som sagts tidigare är

Namn	**Kretstillverkare**
”apm”	Alliance ProMotion och AT-chip.
”ark”	ARK Logic (Advanced Rendering Kernel)
”ati” eller ”atimisc”	ATI (Allied Telesyn International), äldre kort. Nyare kort använder Rage eller Radeon-chip, se nedan.
”chips”	Chips and Technologies ctXXXXXX-chip.
”cirrus”	Cirrus Logic
”cyrix”	Cyrix och Natsemi Geode, företaget numer uppköpt av VIA.
”fbdev”	Linux Framebuffer — egentligen inget grafikkort, drivrutiner i Linuxkärnan används istället.
”glint”	GLINT Permedia, Delta och Gamma-chip.
”i128”	Number 9 I128
”i740”	Intel i740 — används mycket på moderkort med inbyggt grafikkort.
”i810”	Intel i810 och uppåt (i815, i830...)
”mga”	Matrox — klarar de flesta kort, men Matrox tillhandahåller även drivrutiner separat, som ibland är bättre och nyare än X.org-rutinerna.
”neomagic”	Neomagic MagicGraph
”nsc”	National Semiconductor GEODE
”nv”	NVIDIA (klarar inte alla kort och är inte lika bra som företagets egna drivrutiner för X.org-servern)
”r128”	Allied Telesyn International (ATI) Rage 128
”radeon”	ATI Radeon — ATI tillhandahåller egna och ofta bättre drivrutiner till vissa nyare kort, liksom det fristående projektet GATOS, vars rutiner långsamt integreras med X.org-servern.
”rendition”	Rendition eller Micron Verite V1000-, V2000-, V3000-chip.
”s3”	S3 — äldre grafikkort.
”s3virge”	S3 ViRGE och Trio-chip.
”savage”	S3 Savage, ProSavage och Twister-chip.
”siliconmotion”	SiliconMotion Lynx-chip.
”sis”	SiS (Silicon Integrated Systems) chip. Denna drivrutin till X.org-servern (liksom motsvarande framebuffer-drivrutin i Linuxkärnan) är skriven av den svenskspråkige österrikaren Thomas Winischhofer.
”tdfx”	3dfx-chip av alla slag, Voodoo-kort använder dessa.
”trident”	Trident Blade, Image, ProVida, TGUI o.s.v.
”tseng”	Tseng Labs
”vesa”	Klarar alla grafikkort som uppfyller VESA-specifikationen. Denna drivrutin kommer också automatiskt att användas ifall du inte specificerar någon drivrutin alls.
”vga”	Klarar gamla VGA-kort. Nästan alla kort kan hantera detta läge, men de flesta klarar också VESA, så VGA skall bara användas på riktigt gamla kort. Denna ger dock bara 8-bitars färg vilket sällan blir särskilt vackert.
”via”	VIA Technologies (taiwanesiskt företag) grafikkretsar.

Figur 6.10: Drivrutinnamn och motsvarande chipset i X.org-servern.

```
Section "Device"
        Identifier  "Foobar"
        Driver      "foo"
        VendorName  "Foobar International"
        BoardName   "Foobar FunCard"
        VideoRam    8192
EndSection
```

Figur 6.11: Definition av en drivrutin för ett Foobar FunCard-grafikkort med 8 MB videominne.

det enklaste sättet att se vilka sådana som finns att köra **Xorg -configure** och titta på den konfigurationsfil som skapas.

Om detta inte fungerar, eller om du omöjligen kan förstå alla *Option*-inställningar, kan du prova med att skriva t.ex. **man mga**, för att i bästa fall få upp en manualsida om Matroxkorten. Fungerar inte detta är bästa tipset att titta på internethemsidan för X.org-servern, och därnäst att läsa källkoden eller fråga på någon av projektets epostlistor.

Monitor är sektionen som används för att konfigurera din bildskärm.

För detta fordras en hel del information. Normalt finns denna information i manualen till din bildskärm. Om du inte har någon manual, eller har blivit av med den, återstår möjligheten att skaffa fram informationen från tillverkaren eller på annat håll, t.ex. via Internet.

Konfigureringskommandot **Xorg -configure** kommer att försöka fråga bildskärmen vad den kan hantera genom att använda en VESA-standard som kallas EDID, *Extended Display Information Data* som innebär att grafikkortet använder VGA-kabeln till bildskärmen som en kommunikationslänk för att hämta information om den. Det är inte alla grafikkort (speciellt inte äldre kort) som kan hantera denna VESA-standard, och då kan det ibland hjälpa att flytta bildskärmen till en annan dator med ett bättre grafikkort och köra **Xorg -configure** där för att få fullständig information om bildskärmen.

Ett annat bra sätt är att titta i den hårdvarudatabas med namnet *hwdata* som medföljer Fedora Core (och Red Hat): i */usr/share/hwdata* finns en fil som heter *MonitorsDB* och som innehåller data om flera tusen bildskärmar. (Paketet *hwdata* finns även till Debian GNU/Linux och kan installeras med **apt-get install hwdata**.)

Om du åtminstone har tillverkarnamn och modellnummer på din bildskärm brukar den gå att hitta i denna fil.

Följande värden skall konfigureras för en bildskärm:

Identifier (sträng) ett namn för att den *Screen*-sektion som använder bildskärmen skall kunna hitta den.

VendorName (sträng), skall innehålla namnet på tillverkaren, t.ex. **"Foobar"**.

ModelName (sträng), skall innehålla namnet på bildskärmsmodellen, t.ex. **"Foobar Megamonitor 11-P"**. Varken denna eller den föregående variabeln fordras rent tekniskt (och det spelar följdaktligen ingen roll om du stavar fel), utan är avsedda att öka läsbarheten och kvaliteten på eventuella felmeddelanden.

DisplaySize (heltal) (heltal) är två siffror som anger storleken på bildytan i millimeter, $X \times Y$. Du kan mäta med en linjal om du inte redan känner till dessa dimensioner. Siffrorna används för att beräkna skärmens DPI-värde.[24]

HorizSync (flyttal) - (flyttal) anger i vilken hastighet skärmen klarar av att svepa från vänster till höger. Anges antalet sådana svep per sekund, erhålls en frekvens, normalt i kHz-området. Detta kallas *horisontell synkroniseringsfrekvens* och terminologin är präglad av att den ursprungligen har utformats för katodstrålerör (engelska: CRT Cathode Ray Tube) av den typ som finns i TV-apparater och många bildskärmar.[25]

Detta värde skall skrivas in som ett flyttal med bindestreck mellan, och kommer att tolkas som kilohertz (10^3) t.ex.:

```
HorizSync  24.0 - 50.0
```

VertRefresh (flyttal) - (flyttal) anger samma typ av frekvens för den vertikala synkroniseringen, och detta värde faller normalt inom Hz-området och angivet värde kommer att tolkas som Hz, t.ex.:

```
VertRefresh  50.0 - 60.0
```

[24]DPI står för *dots per inch*, d.v.s. punkter per tum, och X- och Y-värdena används t.ex. för att återge cirklar och liknande så att de blir proportionerliga och inte äggformade.

[25]För dagens platta bildskärmar kanske denna terminologi känns lite mossig, men datan överförs faktiskt till dem på samma vis, även om det kanske inte är säkert att bilden faktiskt uppdateras så snabbt som informationen till skärmen överförs.

Gamma (flyttal) (flyttal) (flyttal) anger gammakorrigering för röd, grön och blå färgkanal. Värden mellan 1.0 och 10.0 accepteras. Om du inte arbetar professionellt med grafik eller har en aning om vad detta är kan du lugnt ignorera det hela och strunta helt i denna rad.

När en bildskärm visar en viss intensitet av en viss färg (röd, grön eller blå) kommer den att förvränga signalen från grafikkortet något, så att bara min- och maxintensiteten för färgen visas korrekt. Kurvan från intensitet 0.0 till 1.0 är alltså inte lineär. För att kompensera för detta kan vissa grafikkort utföra gammakorrigering på den elektriska signal som skickas till bildskärmen. Att mäta upp dessa korrigeringsvärden exakt kräver normalt speciella instrument som appliceras på bildskärmen.

Om bilder på din skärm har en lätt missfärgning jämfört med originalen, kan det vara så att du har behov av gammakorrigering. Är det rödstick i bilden, så skall gammakorrektionsvärdet för grönt och blått ökas för att kompensera för detta, o.s.v. Du kan testa fram gammakorrektionsvärden interaktivt med programmet **xgamma** (se nedan), men för att göra ändringen permanent krävs att du skriver in den i konfigurationen för bildskärmen.

Option ”DPMS” finns nästan alltid satt på moderna bildskärmar, och dess värde för en okänd skärm framgår av *hwdata*-filen från Fedora.

DPMS står för *display power management signaling* och är en standard utformad av VESA för att signalera till en bildskärm att den skall ”blanka” d.v.s. gå ned i viloläge och släcka skärmen. Detta skall normalt vara påslaget om din monitor stödjer det, eftersom det sparar både energi, pengar (även livstiden för bildskärmen förlängs) och miljö.

En fullständig *Monitor*-sektion kan ha ett useende som påminner om figur 6.12.

Utöver denna konfiguration *kan* sektioner med namnen *ServerFlags*, *VideoAdaptor*, *Modes* och *Vendor* förekomma. De har dock sällan synts till i det verkliga livet, så du kan nog utgå från att du slipper att se dem.

```
Section "Monitor"
        Identifier   "Bar"
        VendorName   "Foobar"
        ModelName    "Foobar Megamonitor 11-P"
        DisplaySize  290        210
        HorizSync    24.0 - 50.0
        VertRefresh  50.0 - 60.0
        Option       "dpms"
EndSection
```

Figur 6.12: Definition av en billig bildskärm med den fysiska storleken 290 × 210 millimeter, och stöd för energisparläge enligt DPMS.

6.3 Programmen i X

Även om X.org-servern för det mesta ”bara finns” följer det faktiskt med en hel del program i paketet, som kan användas för att administrera, testa och finjustera *X*. Här följer en lista över de viktigaste, mest basala *X*-programmen:

Xorg eller bara **X** är själva huvudprogrammet, *X*-servern. Det är detta program som läser in konfigurationsfilen, initierar musar och grafikkort, och visar grafik under GNU/Linux.

Under förutsättning att ingen annan *X*-server redan kör i ditt operativsystem kan du faktiskt starta X.org-servern med bara kommandot **Xorg** eller rent av bara **X** följt av enter. Du får då en svart *X*-skärm med en *X*-formad markör. Du *kan* starta program på denna arbetsyta manuellt, t.ex., om du växlar till en annan textkonsoll och skriver:

```
# export DISPLAY=:0
# xeyes&
```

Detta kommer att tala om för programmen att de skall startas på skärm 0 på den lokala maskinen. När du växlar tillbaka till *X* med **Ctrl+Alt+F7** ser du att programmet har startats.

Du kan starta flera *X*-servrar på en och samma dator om du har lust. **Xorg :1** kommer att starta en andra skärm med numret 1, och lägger den i VT8, så att du når den med **Ctrl+Alt+F8**. Såhär kan du hålla på om du vill.

Eftersom det är ganska tråkigt med en sådan här svart *X*-server finns det andra program som kan starta X.org-servern på ett lite mer intuitivt vis.

xinit startar en enkel *X*-server och kör sedan en skriptfil som heter */etc/X11/xinit/xinitrc*, som först konfigurerar en del grundläggande saker och sedan i sin tur skall se efter om användaren har någon *.xinitrc*-fil i sin hemkatalog. Om en sådan finns skall denna att köras som om den vore ett skript.

Om ingen sådan skriptfil finns, händer normalt något annat; det vanligaste är att systemet startar en standardiserad skrivbordsmiljö eller en mindre fönsterhanterare. Det är alltså härifrån som GNOME eller KDE startas, antingen direkt eller via ett eller annat sekundärt skript.

Notera alltså, att om du vill byta ut fönsterhanteraren på ett POSIX-system som kör *X*, så skall du som användare kunna göra detta genom att lägga in en *.xinitrc*-fil med ett skript. Du kan därför installera en hel fönsterhanterare i din hemkatalog och köra den helt enkelt genom att ändra i *.xinitrc*. En fönsterhanterare skall alltid startas med kommandot **exec**, t.ex. **exec twm**. Den mycket spartanska fönsterhanteraren TWM (Tiny Window Manager) följer med X.org-servern och finns oftast tillgänglig för den som vill skala av allt grafiskt lullull från sin fönsterhanterare med en *.xinitrc* i den här stilen (TWM har en svart bakgrundsskärm och *X*-formad markör, du måste trycka på vänsterknappen för att något skall hända efter uppstart):

```
#!/bin/sh
exec twm
```

Samtidigt är det värt att påpeka, att om du nu skulle göra något fel, så att det som startas i din *.xinitrc*-fil inte fungerar, finns det risk att du skjuter dig själv i foten och låser dig ute från fönstersystemet. Då kan du bli tvungen att logga in via en textterminal från en annan dator för att rätta till felet, eller bara ta bort den felaktiga *.xinitrc*-filen.

Det är faktiskt t.o.m. möjligt för en användare att byta ut själva *X*-servern genom att använda skripfilen *.xserverrc* i sin hemkatalog, och därifrån köra önskad server med skalkommandot **exec**. Detta får dock betraktas som mycket ovanligt.

startx är det kommando som brukar föredras för att starta *X* efter inloggningen. Det använder i sin tur **xinit**, så detta kommando är egentligen bara till för att förenkla hanteringen av föregående kommando. En vanlig variant är:

startx -- -depth 8 starta *X* med bara 8 bitars färgdjup (under förutsättning att ditt grafikkort kan hantera det och att en motsvarande *Display*-undersektion existerar i din *Screen*-sektion i xorg.conf). **-depth 16** går t.ex. också bra för att testa om prestanda ökar i detta färgläge. De två bindestrecken som föregår parametern talar om att det är *X-servern* som skall påverkas.

xdm — *X Display Manager* är en standardiserad *skärmhanterare*. (Förväxla ej med *fönsterhanterare*!).

En skärmhanterare är ett demonprogram som när det första gången startats i sin tur startar **xinit**, och instruerar *X*-servern att starta ett inloggningsfönster, vanligtvis bara för inloggning på den egna datorn. När en användare sedan loggar ut stängs *X*-servern av, och XDM kommer då att upptäcka detta och starta en ny inloggningsskärm, så att nästa användare kan logga in på datorn. De flesta GNU/Linux-system är konfigurerade att fungera på just detta vis.

Projekten GNOME och KDE har skrivit egna skärmhanterare, **gdm** respektive **kdm** som utför samma sak som XDM, med skillnaden att de har lagt till en del färg och form, språkval, samt några extra funktioner för att t.ex. stänga av datorn från inloggningsskärmen utan att logga in.

I ett större system med många datorer, där användare skall kunna logga in på varandras maskiner, eller där du vill kunna använda tunna klienter, s.k. *X*-terminaler[26] kan XDM användas så, att det istället för inloggningsskärmen presenterar en lista på tillgängliga datorer, ur vilken du väljer en dator som du vill logga in på, och sedan får upp den vanliga inloggningsrutan fast *på den valda datorn*. Hela din session kommer sedan att köras på den andra datorn, medan grafiken skickas över nätverket.

Eftersom **xdm** och liknande program används av de flesta distributioner kan det uppstå problem när du t.ex. vill ladda i en ny konfiguration i X.org-servern. Det kan då vara nödvändigt att ta ner maskinen i en icke-grafisk körnivå med **init 4** och sedan starta X-servern igen med **init 5**.

Det är viktigt att se till att du har en fungerande *X*-server innan du sätter den till att starta varje gång du går upp i körnivå 5, om detta är den nivå maskinen automatiskt startar i, vilket det normalt är. Saken är den, att init-demonen nämligen inte ger sig förrän den

[26]Se vidare sidan 256.

får igång det som angetts i varje körnivå, så om X-servern kraschar så startas den om (kallas "respawn") *ad nauseam* tills dess den lyckas eller tills init-demonen upptäcker att den faktiskt bara startar om hela tiden. Detta gäller alla program du startar från *inittab*-filen med "respawn", men är ovanligt störande när det gäller *X*, eftersom skärmen och konsollerna flimrar och det blir svårt att kontrollera systemet över huvud taget. Du kan bli tvungen att starta om datorn och dra upp kärnan i enanvändarläge för att sedan editera */etc/inittab* så att inte *X*-servern startas om hela tiden. (Se även avsnitt 5.4.3 på sidan 189 om *init* och körnivåer.)

xdpyinfo (kortform av *X display information*) visar grafikprestanda för den skärm du för tillfället använder. Då menas inte "bildskärm" utan det som svarar mot *Screen*-sektionen i konfigurationsfilen, d.v.s. kombinationen av grafikkort, drivrutin och bildskärm. Detta tolkas oftast bara som "X-serverns prestanda". Med prestanda menas i detta fall hur stor upplösning skärmen har, hur stor markören kan vara, diverse teknisk information samt vilka *Visuals* skärmen kan hantera.

En *Visual* (något bra svenskt ord har jag inte hittat) är en beteckning på ett visst färghanteringsläge. Nuförtiden betyder det för det mesta *DirectColor* eller *TrueColor*, vilket innebär att varje bildelement har reserverat t.ex. 24 bitar i grafikminnet för att välja en godtycklig färg.

Gamla grafikkort stödde normalt bara *PseudoColor*, vilket betydde att färgen på en pixel inte sattes individuellt med ett antal bitar per pixel, utan att de 8 bitarna i en pixel användes för att peka ut en färg ur en viss *palett*, som sedan i sin tur ändrades. Om färgen i paletten byttes ut, byttes därför färgen på alla bildelement som refererade till denna palettposition. Många programmerare använde detta trick, som inte fungerar med moderna grafikkort annat än i 8-bitarsläget, och därför finns det ännu en del program som klagar över att de inte kan hitta PseudoColor på datorn.[27]

xset kan ställa in en del variabler i *X*-servern, det allra vanligaste användningsområdet torde vara **xset b off** vilket stänger av det evinnerliga pipandet som uppstår när du t.ex. bläddrar högst upp i en fil och inte kommer längre. Följande brukar användas mer eller mindre sällan:

---

[27] Se avsnitt 6.2 på sidan 239 för information om hur du aktiverar PseudoColor för ditt grafikkort.

Kommando	**Effekt**
xset b off	Stänger av terminalpipet.
xset b on	Slår på det igen.
xset c off	Slår av terminalklicket.
xset c on	Slår på det igen.
xset -dpms	Slår av DPMS energisparfunktioner (se sidan 245)
xset +dpms	slår på det igen.
xset r off	Stänger av tangentrepetition. (Att hålla nere en tangent genererar inte en lång rad tecken, utan bara ett enda.)
xset r on	Slår på det igen.
xset q	rapporterar aktuella inställningar.

xhost används för att ange vilka datorer som får, respektive inte får ansluta till din *X*-server. Det anses att den som arbetar vid *X*-servern skall avgöra vilka andra datorer som skall tillåtas att öppna fönster på den.

Med **xhost +** tillåts *alla* datorer i *hela världen* att öppna fönster på din dator. Det är kanske inte så bra, men du kan vara ganska övertygad om att världen inte kryllar av personer som försöker öppna fönster på andras datorer på pin kiv. Därför är det ganska vanligt att använda detta kommando för att släppa in främmande fönster. Om du är mer försiktig av dig kan du skriva t.ex. **xhost foo.bar.org** för att bara låta anslutningar från datorn med namnet *foo.bar.org* gå fram.

För att öppna ett fönster från den andra datorn räcker det med att sätta miljövariabeln **$DISPLAY** så att den pekar på din dator. Normalt behöver även skärmens ID-nummer anges, t.ex. *foo.bar.org:0*. Om *din* dator heter *fnord.foo.org* skall användaren på datorn med namnet *foo.bar.org* alltså skriva exempelvis:

```
# export DISPLAY=foo.bar.org:0
# xeyes&
```

Detta kan du själv prova från en annan dator om du har ett inloggningskonto där. Om du själv vill öppna fönster från en annan dator hem till dig själv, där du sitter, skall du dock normalt använda SSH, som i de flesta fall automatiskt krypterar och överför en *X*-förbindelse mellan två eller flera datorer. Om detta inte vill fungera kan dock ovanstående vara till hjälp.

setxkbmap gör det möjligt att byta tangentbordslayout. Alla möjliga layouter är representerade i */usr/share/X11/xkb/symbols* men de som en svensk användare kan tänkas behöva är väl:

Namn	**Layout**
"se"	Svenskt standardtangentbord
"dvorak(se)"	Svenskt Dvorak-tangentbord
"sapmi(sefi)"	Samiskt tangentbord för norra Sverige
"dk"	Danskt tangentbord
"fi"	Finskt tangentbord
"no"	Norskt tangentbord

För att byta till samiska skriver du t.ex. **setxkbmap "sapmi(sefi)"**. Någon speciell tangentbordslayout för meänkieli, jiddisch eller romani chib fanns inte när jag undersökte detta, men det är inte särskilt svårt att åstadkomma en modifierad variant om det skulle behövas. Nästa kommando kan faktiskt användas till sådana saker:

xmodmap används för att modifiera tangentbordslayouten, och kan t.ex. byta plats på vissa knappar, men även byta hela layouten i ett svep på samma vis som **setxkbmap**. Du skall dock använda det föregående kommandot för att byta hela layouten; **xmodmap** är ett gammalt *X*-program.

Det vanligaste användningsområdet för en svensk torde väl vara att pilla in svenska tangenter på en dator som påträffats i utlandet, men som inte har **setxkbmap**. Du kan byta en individuell tangent med t.ex. **xmodmap keycode 9 = A** för att få fram ett stort *A* vid tryck på **Esc**-tangenten.

Vilka siffror som betyder vad efter *keycode* är däremot inte så lätt att veta, men i katalogen */usr/share/X11/xkb/keycodes* finns filer som visar hur olika fysiska tangentbord mappar till siffror. Du kan sedan med **xmodmap** välja att mappa dessa siffror till ett visst tecken. Följande sekvens ger svenska tecken på de positioner där vi är vana vid att ha dem:

```
keycode 34 = aring Aring
keycode 48 = adiaeresis Adiaeresis
keycode 47 = odiaeresis Odiaeresis
```

Raderna ovan kan skrivas in i en fil och köras med **xmodmap foo.map** eller köras en i taget med t.ex.: **xmodmap -e "keycode 34 = aring Aring"**.

xgamma är ett interaktivt program för gammakorrigering, en princip som beskrivs på sidan 245.

xvidtune är ett interaktivt program för att justera inställningarna för horisontell och vertikal uppdateringsfrekvens. Det är avsett för den gamla arkitekturen med *Modeline*-rader i konfigurationsfilen, och skall inte användas med X.org-servern annat än om du prompt måste skriva sådana *Modeline*-saker i din konfigurationsfil. Programmet kan orsaka fysiska skador på såväl grafikkort och bildskärm vid felaktigt användande.

xkill kan användas för att välja ut ett fönster och döda det program som hanterar fönstret, så att det försvinner, som en grafisk motsvarighet till POSIX-kommandot **kill**. Eftersom de flesta progam i moderna fönsterhanterare har en stängningsknapp (×) högst upp till höger i fönstret, är detta kommando ganska gammaldags och förlegat.

xsetroot byter utseende på rotfönstret. T.ex. **xsetroot -solid gray** kommer att göra rotfönstret grått istället för svart.

De flesta fönsterhanterare har egna rutiner för att byta detta utseende, och vissa (t.ex. GNOME) lägger ett enda stort, vanligt fönster *över* rotfönstret så att det inte alls syns, och då är det ju ingen större mening med det hela. Använd alltså inbyggda kommandon i skrivbordsmiljön eller fönsterhanteraren för att modifiera detta.

Utöver dessa grundläggande program medföljer också ett antal program för användare, utan vilka det vore ganska svårt att över huvud taget använda *X* till någonting när det väl installerats. De är liksom ovanstående program ursprungligen från systemet *X* och är därför inte utformade för eventuella skrivbordsmiljöer eller fönsterhanterare som du använder. Fördelen med dem är att de alltid finns tillgängliga och kan användas för att testa *X*.

De flesta programmen är svartvita, vilket beror på att de är anpassade för att kunna köras på datorer med 1 bits färgdjup vilket oftast är vitt på svart, men på många äldre datorer kan vara t.ex.terminalgrönt eller terminalgult på svart.

xterm är det mest använda av programmen som följer med *X*. Det används för att ge användare ett terminalfönster under *X*, där sedan användarens standardskal öppnas. I olika skrivbordsmiljöer och fönsterhanterare finns olika versioner av den här typen av terminalfönster, och du bör snabbt lära dig att hitta dem på ett nytt system — men fungerar inget annat kanske du kan klura ut

ett sätt att köra igång det hederliga gamla **xterm**. I vissa POSIX-system är detta rent av standardterminalen.

xconsole ger utskrifter som normalt skulle hamnat på */dev/console*. Prova att starta **xconsole** och i ett terminalfönster skriva:

```
echo "fnord" > /dev/console
```

Normalt skickas denna text till aktuell terminal på datorn, men **xconsole** kommer att fånga upp meddelanden från alla användare och visa dem i ett fönster. I GNU/Linux skickas bland annat alla meddelanden från kärnan till */dev/console*, och dyker då och då upp som ovälkommen text i något terminalfönster.

xload visar ett svartvitt rullande stapeldiagram som visar hur belastad datorn är över tid.[28]

xlogo ger en vacker *X*-logotyp i ett eget fönster. Detta program är föga användbart annat än möjligen till tester.

xcalc ger en mycket enkel och rudimentär miniräknare.

oclock ger en gammal klocka med visare.

xclock ger en modernare klocka med visare, och lite färger om du har tur.

xeyes är ett omtyckt program med två ögon som tittar på muspekaren.

xbiff ger en bild på en nordamerikansk postlåda med en flagga på sidan som far upp om du får epost. Namnet kommer från en hund med namnet *Biff*. Hundar skäller som bekant när brevbäraren kommer. Din dator måste leverera epost lokalt för att detta skall fungera.

6.4 Teckenuppsättning

Hanteringen av teckenuppsättningen i *X* är en ganska udda historia som ibland kan dyka upp i de mest oväntade sammanhang, och som därför behöver förklaras närmare.

Fonterna (vilket är det engelska ordet för *stilsats*, d.v.s. ett samlingsplats för ett typsnitt, och som idag brukar beteckna den fil i vilken ett

[28]Är du intresserad av den här typen av belastningsövervakning rekommenderas ett modernare program, t.ex. **gkrellm**.

teckensnitt finns lagrat) i X.org-servern hittas genom att söka i de sökvägar som angivits i sektionen *Files* i någon av variablerna för *FontPath*.

Den del av X.org-servern som hanterar fonter kallas *Xfont* och använder numera programbiblioteken *FreeType* och *Fontconfig* för att utföra jobbet. Fontconfig hanterar inläsningen av fonterna, medan FreeType hanterar visningen.[29] FreeType ger stöd för följande typer av fonter:

TrueType (*.ttf, *.ttc)	Type 1 (*.pfa, *.pfb)
CID-kodad Type 1	CFF
OpenType	SFNT-bitmapfonter (*.snf, *.bdf, *.pcf)
X11 PCF	Windows FNT
BDF	PFR
Type42	Bitstream Speedo (*.spd)

Du kan lägga till fonter för hela *X*-systemet genom att lägga dem i någon av katalogerna under */usr/share/X11/fonts*. Därefter måste du köra kommandot **fc-cache** för att X.org-servern skall upptäcka dem.

En enskild användare kan också lägga till fonter genom att lägga dem i den dolda katalogen *.fonts* i sin hemkatalog, och de kommer då att läsas in nästa gång användaren loggar in. S.k. TrueType *.ttf*-fontfiler från Microsoft Windows kan t.ex. användas utan problem. De kommer då att integreras med fontsystemet i *X*.

Namnen på fonter i *X* är något förskräckande, t.ex.:

```
-misc-fixed-*-*-*-*-20-*-*-*-*-*-*-*
```

Detta beror på att en applikation som vill använda en viss teckenuppsättning bara specificerar vissa *krav* på vad som önskas. Stjärnorna (*) ovan har står för ”vad som helst”, precis som om de vore ett globbningsuttryck. De 14 fälten specificerar 14 möjliga önskemål, från vänster till höger:

Foundry betyder egentligen *gjuteri*, och betecknar upphovsman till fonten. Förr i tiden gjöts ju en font av någon i form av blytyper, därför kallas upphovsmannen till en font gjutare. Flera företag har donerat fonter till *X*-projektet.

Family betecknar en viss familj av fonter, t.ex. *times* eller *courier*. Ordet *sans* som förekommer i en del familjer betyder *sans serif*, d.v.s.

[29] Egentligen är det *X* ↔ Xfont/Xrender ↔ Xft ↔ FreeType/Fontconfig. Webbläsaren Mozilla, ordbehandlaren OpenOffice.org samt skrivbordsmiljöerna GNOME och KDE använder också FreeType och Fontconfig, ibland genom *Xrender* och ibland inte. I framtida X-servrar såsom Xgl används antagligen inte Xrender alls, utan OpenGL används direkt.

"utan serifer" — att fontens tecken saknar de för Times och liknande utstickande klackarna upptill och nedtill på tecknen som anses underlätta läsningen.

Weight betecknar teckentjockleken hos tecknen, t.ex. *bold* (fetstilt).

Slant betecknar hållningen på tecknen, *i*talic (kursiv) *r*oman (romansk) eller *o*blique (både och).

Set width är ett medelvärde på bredden av tecknen i fonten. Eftersom tecknen kan vara olika breda för olika bokstäver används detta värde för att välja genomsnittlig bredd. Det är inte siffervärden som anges, utan ett antal fördefinierade "breddegenskaper", t.ex. *bold* (fet), *normal* eller *condensed* (förtätad).

Additional stye väljer ytterligare egenskaper hos fonten. Här är det möjligt att välja mellan två eller flera varianter av en font, t.ex. *sans* för en variant utan serifer.

Pixel size Z höjden på själva fonten i bildelement. Precis som på en blytyp inkluderar höjden eventuellt vitt utrymme under och över själva bokstäverna.

Point size Z Samma höjd, fast angiven i antal punkter, en gammal typografisk måttenhet. En punkt $\approx \frac{1}{72}$ tum $\approx 0,353$ millimeter.

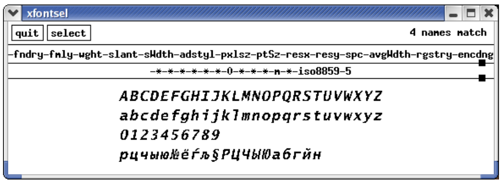
Resolution X den skärmupplösning i X-led som fonten är avsedd för, angivet i bildelement per tum.

Resolution Y den skärmupplösning i Y-led som fonten är avsedd för, angivet i bildelement per tum.

Spacing är den typ av mellanrum som används i fonten angivet med en bokstav, där *m*onospaced är den enklaste typen: alla tecken är lika breda, vilket krävs av t.ex. terminalfönster, *p*roportional tillåter att ett *i* är betydligt smalare än ett w precis som i den här bokens text.

Average width den genomsnittliga bredden av tecknen i fonten angivet i antal tiondels bildelement.

Registry är namnet på den organisation eller standard som är upphov till kodningen av fonten, d.v.s. i vilken ordning de olika tecknen kommer. För unicode är det *iso10646* som gäller, medan den gamla ISO-standarden har namnet *iso8859*.



Figur 6.13: X font selector (**xfontsel**) kan interaktivt testa olika fonter i *X*.

Encoding kodningen för denna teckenuppsättning. Detta värde utnyttjas normalt tillsammans med föregående, för att välja en ISO 8859-standard, sätt t.ex. **Registry** till *iso8859* och **Encoding** till *5* för att välja teckenkodningen *iso8859-5* (kyrilliska tecken). Används *iso10646* i **Registry** skall detta vara satt till *, eftersom unicode per definition innehåller *alla* tecken.

Vissa program (t.ex. orbehandlaren OpenOffice.org) förenklar denna hantering avsevärt, för att göra det lite mer bekvämt för användaren.

För att välja ut ett teckensnitt genom att ange dessa värden är ganska tidsödande. Det finns ett antal program som hjälper till med valet:

xlsfonts (utläses *list X fonts*) listar alla tillgängliga fonter i systemet. Det blir ganska många.

xfontsel (*X font selector*) är ett program för att interaktivt testa att välja en teckenuppsättning genom att skruva på ovanstående parametrar. Detta följer alltid med *X*, se figur 6.13.

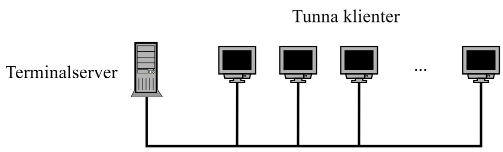
gfontsel (*GNOME font selector*) är den senare GNOME-varianten av samma program, som ibland finns tillgängligt.

fc-list är ett program som hör samman med programbiblioteket Fontconfig, och som listar de fonter som just Fontconfig håller reda på. Fontconfig väljer att skala av en hel del av de detaljparameterar som *X* normalt använder.

fc-cache uppdaterar som nämnts Fontconfig:s lista över tillgängliga fonter.

6.5 X-terminaler

Startpunkt: `http://www.ltsp.org/`



Figur 6.14: X-terminaler ansluts över nätverk till en central dator som tillhandahåller all programvara utom X-servern och terminalens operativsystem.

Att använda s.k. *X-terminaler* eller *tunna klienter* som de också kallas, har blivt allt mer populärt. Flera svenska företag säljer lösningar som har till mål att byta stora, dyra datorsalar på t.ex. bibliotek och skolor, mot billiga datorsalar med tunna X-terminaler.

Med *tunn klient* menas helt enkelt en klen dator. Så klen är den tunna klienten, att den bara förtjänas att kallas terminal, precis som de gamla terminalerna av den typ som omtalas i avsnitt 2.3.11 på sidan 105 och framåt. (Repetera här lämpligen även stycket om klient och server i avsnitt 2.3.10 på sidan 104.)

X-terminalerna fungerar så, att den klena datorn förses med ett mycket kompakt och rudimentärt GNU/Linux-system, som nätt och jämnt är kapabelt att starta upp *X*-servern. Själva operativsystemet behöver inte ens ligga på en hårddisk i själva datorn, utan kan laddas ned från nätverket.

X-servern på denna terminal konfigureras sedan för att antingen öppna en skärm direkt mot en annan dator, eller för att använda programmet XDM (se sidan 248) som via protokollet XDMCP (X Display Manager Configuration Protocol) kan hämta en lista över datorer som finns tillgängliga för inloggning, om det nu finns flera tillgängliga. Den dator som tillhandahåller tjänsten kallas *terminalserver*.

På terminalservern, som är en så kapabel arbetshäst som möjligt, med mycket minne och fillagringsutrymme, finns sedan ett fullständigt utbyggt GNU/Linux-system med skrivbordsmiljö och allt annat som ingår. Allt detta upplever användaren som lika snabbt som om den egna X-terminalen var en pigg och alert dator, trots att den bara har till uppgift att hantera grafik, mus och tangentbord.

Det är inte särskilt svårt att inse att detta är exakt den ursprungliga avsikten med MIT:s Athena-projekt som berättades om i början av detta kapitel: att använda billig utrustning för att minska datordriftkostnader. Många använder också gamla datorer som blivit stående som X-terminaler, eftersom många sådana maskiner klarar väl av att visa grafik och kommunicera över nätverk i hög hastighet.

Kommersiella leverantörer förser ofta sina X-terminaler med möjlighet att även köra Microsoft Windows-program, eller att ansluta till Windows terminalservrar, eftersom dessa är mycket populärta.

Ett problem med X-terminaler är att andra delar av maskinvaran, bortsett från grafik och inorgan, inte är nätverkstransparenta. T.ex. är det knepigt att få ljudet i GNU/Linux att transporteras över nätverk, och t.ex. USB-portar och liknande som finns på en "riktig" dator är oftast stört omöjligt att ansluta till en billig X-terminal. De passar därför bäst i mindre avancerade miljöer, t.ex. för vanliga kontorsgöromål eller undervisning.

Många företag väljer att bygga sina egna tunna klienter med utgångspunkt från t.ex. *Linux From Scratch*, men de flesta sneglar ändå på LTSP (Linux Terminal Server Project) som har en färdig lösning som fungerar tillsammans med de flesta distributioner, och detta är den startplats som rekommenderas för att komma ingång med X-terminaler på tunna klienter.

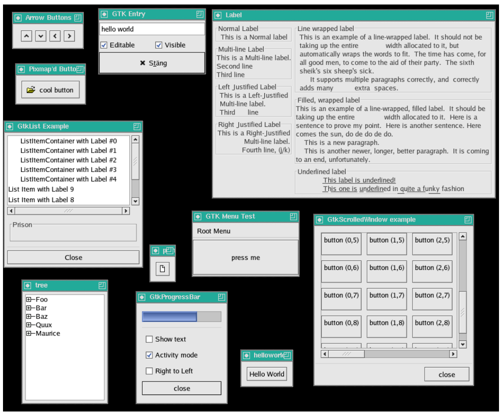
6.6 Skrivbordsmiljöer

Skrivbordsmiljöerna är den del av GNU/Linux som utvecklats mest explosivt under de senaste åren. Detta hade sin grund i att det avskalade *X* och dess simpla fönsterhanterare är så primitiva och ointuitiva att de faktiskt är obegripliga för en s.k. "vanlig användare". Därför har två separata projekt startat som intimt förknippas med den grafiska miljön i GNU/Linux: KDE och GNOME.

Dessa skrivbordsmiljöer är så utformade att de, på samma vis som skrivbordsmiljöerna för Macintosh eller Microsoft Windows, skall erbjuda användaren allt denne kan önska sig av en modern skrivbordsmiljö på en dator.

Skrivbordsmiljöerna är också *utvecklingsplattformar* eller *ramverk*: en utvecklare som skriver ett program för GNOME eller KDE, vet att programmet kommer att kunna användas av alla operativsystem som kan använda GNOME eller KDE. Det behövs sällan särskilt många anpassningar av ett program för att lyckas med detta. Skrivbordsmiljöerna innehåller s.k. *ramverk* för programutveckling, vilket betyder att det finns en mängd färdiggjorda saker som kan återanvändas.[30]

[30] Många tidiga program som skulle användas i flera olika operativsystem var tyvärr tvungna att uppfinna det här hjulet på nytt. Program som OpenOffice.org eller Mozilla bär tydliga tecken av detta; båda programmen har egna portabla grundtyper, trådabstraktion, widget:ar, komponentobjekt etc. som behövs för skrivbordsprogram. Både GNOME och KDE erbjuder detta färdigt.



Figur 6.15: Några av de widget:ar (manicker) som tillhandahålls av GTK+.

För en historisk tillbakablick på skrivbordsmiljöerna för POSIX-system börjar vi dock med CDE.

6.6.1 Motif och CDE

I begynnelsen fanns Athena-projektet, och inte ens de orkade använda det mycket krångliga programmeringsgränssnittet i *X*. Av den anledningen byggde MIT en uppsättning *widget:ar*. Ordet *widget*, som på svenska blir *grej*, *manick* eller *liten pryl*, betecknar ett grafiskt byggelement av den typ som krävs för att kunna bygga grafiska användargränssnitt. Vi är här tvungna att använda den något klumpiga försvenskningen *widget*, eftersom det inte finns något vedertaget svenskt ord för detta.

Widget:ar är t.ex. inkapslingsboxar, knappar, menyrader, listboxar, radioknappar, kryssboxar, trädstrukturer, o.s.v. Figur 6.15 visar några widget:ar från GTK+. Widget:ar brukar sägas vara grunden till ett fönstersystems "look and feel": du ser direkt om en skärmbild är tagen från GTK+, Qt- eller för den delen Windows API.

En widget-uppsättning brukar förutom dessa byggelement också innehålla ett system för att hantera *händelser*, varmed avses t.ex. tryck på musknappar och tangentbord. Program som använder detta kan registrera sig som "lyssnare" på vissa händelser, så att de på det viset kan snappa upp signaler från användaren och antingen blockera och behandla dem, eller skicka dem vidare till andra komponenter i systemet.

De widget:ar som Project Athena skapade kallades *Xaw*, efter filen där de låg, *libXaw.a*, och det skall förmodligen förstås som *Athena widgets for X* eller något liknande. Dessa användes under ganska lång tid, eftersom de distribuerades tillsammans med *X*.

Runt år 1989 hade *X* blivit så mycket standard i UNIX-världen att organisationen Open Software Foundation såg det som nödvändigt att standardisera den allt mer spretiga arbetsmiljön under *X*. Därför skapades widget-uppsättningen *Motif* som senare standardiserades som IEEE 1295, och i förlängningen av Motif *CDE*, *Common Desktop Environment*, den gemensamma skrivbordsmiljön för alla UNIX-system.

Med Motif följde en stilmall och riktlinjer för hur applikationer skulle utformas för att vara giltiga Motif-applikationer. Motif vandrade sedan tillsammans med CDE fram och tillbaka mellan olika organisationer, och tillhör idag The Open Group. Motif eller CDE har dock aldrig någonsin varit fri programvara, utan alltid krävt godkännande av en proprietär licens.

CDE tillkom inte förrän 1993, då ett antal leverantörer (bl.a. Sun Microsystems, Hewlett-Packard, IBM, Novell o.s.v.) annonserade att de tillsammans skulle utveckla Common Open Software Environment (COSE), varav den gemensamma skrivbordsmiljön CDE skulle vara en vital del.

Efter detta blev CDE med Motif den standardiserade skrivbordsmiljö som används i alla stora, proprietära, grafiska UNIX-system. Detta höll i sig tills augusti år 2000 då Sun Microsystems, den förmodligen viktigaste drivkraften bakom CDE, annonserade att de skulle överge CDE till förmån för GNOME.

Vad CDE tillförde var:

- Sessionshantering — tillståndet på skrivbordet kunde lagras.
- Virtuella skärmar — det var nu möjligt att ha flera mindre arbetsytor av ett större virtuellt skrivbord.
- Frontpanelen — en panel med möjlighet att starta applikationer av den typ som ofta finns längst ned eller längst upp i moderna grafiska operativsystem.
- En filhanterare och en applikationshanterare för att navigera runt bland filer och starta applikationer.
- Ett antal standardverktyg: texteditor, ikoneditor, online-hjälpsystem, en bättre terminalemulator än gamla **xterm** och en miniräknare, till exempel.
- En utskriftshanterare.

- Drag-och-släpp-funktionalitet mellan applikationer.
- Internationalisering och internationellt språkstöd.

Det är fascinerande att se hur mycket av detta som kännetecknar de moderna skrivbordsmiljöerna: i princip alla dessa punkter är fortfarande den viktiga kärnan i alla sådana miljöer.

När CDE kom runt 1994 var det främst inspirerat av Microsoft Windows 3.11, men till skillnad från Windows har det inte utvecklats många steg sedan dess. CDE ser idag i princip likadant ut som då. Rent estetiskt skulle nog många formgivare hävda att CDE är fult som stryk, men det är nog snarare tidstypisk gränssnittsdesign från mitten av 1990-talet. Inte heller det underliggande Motif har förändrats nämnvärt.

6.6.2 Qt och KDE

Officiell hemsida: `http://www.kde.org/`

År 1994 hade osloborna Eirik Eng och Haavard Nord bildat företaget *Quasar Technologies* i syfte att utveckla en ny uppsättning widget:ar i stil med Motif. Skillnaden här var att denna widgetuppsättning, som fick namnet *Quasar Toolkit*, eller kort och gott *Qt*, skulle gå att använda oavsett om programmen var skrivna för POSIX-operativsystem, Macintosh eller Microsoft Windows.

Företaget Quasar Technologies bytte sedan namn till *Trolltech*, vilket gör att namnet Qt kanske är lite maplacé, men det har ändå fått hänga med.

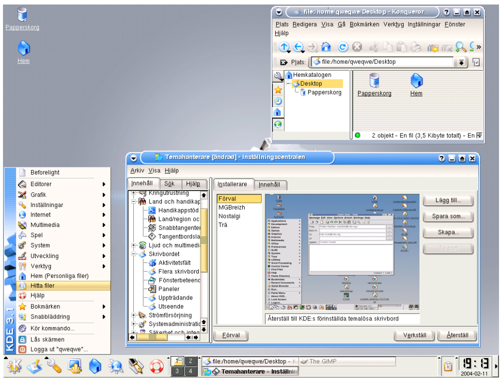
När sedan tysken Matthias Ettrich[31] till slut tröttnade på att inte kunna använda någon ordentlig skrivbordsmiljö under GNU/Linux, och därför bestämde sig för att skriva en egen, valde han att basera denna på Qt. Till skillnad från många andra widget-uppsättningar var Qt skrivet för det objektorienterade programspråket C++ vilket ytterligare tilltalade Ettrich.

I en postning till ett antal Linuxinriktade nyhetsgrupper i oktober 1996 utlyste han projektet *Kool Desktop Environment*, KDE. Förutom att be om hjälp med programmeringen, skissade han även upp projektets syfte. Där framgår bland annat:[32]

> Tanken är *inte* att skapa ett grafiskt användargränssnitt för hela UNIX-systemet eller för systemadministratören. För det ändamålet är UNIX kommandoradsgränssnitt med tusentals verktyg och skripspråk mycket bättre. Idén är att skapa

[31] Ettrich är annars mest känd för att ha skrivit LaTeX-editorn LyX.

[32] Min översättning.



Figur 6.16: K Desktop Environment (KDE) har ett standardtema som lätt känns igen. Ett annat kännetecken är en förkärlek för kugghjul.

> ett grafiskt användargränssnitt avsett för en *slutanvändare*. Någon som vill surfa på nätet med Linux, skriva några brev och spela några trevliga spel.[9]

Flera utvecklare hörsammade omedelbart Ettrich, och började utveckla KDE tillsammans med honom. En av dem var Matthias "Kalle" Dalheimer i Värmland.

På det fåtal år som gått sedan denna annons har KDE utvecklats till en fullständig skrivbordsmiljö. Från början var syftet att nå upp till och överglänsa CDE, vilket snart lyckades. Sedan sattes fokus på att nå samma stil och användarvänlighet som Microsoft Windows, något som också måste sägas ha lyckats. KDE är idag känt för att vara det normala förstahandsvalet för GNU/Linux-användare som kommer från Microsoft Windows-världen.

För att utforma skrivbordssystem krävs nästan alltid någon form av komponentobjektmodell. Dessa är en utveckling av grundtanken som framkastades av Douglas McIlroy redan 1968, men skillnaden mellan *rör och filter*-modellen är att komponentobjekt kan kommunicera i två riktningar. En anropad komponent kan svara på ett meddelande och lämna upplysningar som den anropande komponenten tar hänsyn till.

I skrivbordssystem är komponentobjekt speciellt viktigt, eftersom drag-och-släpp-funktionalitet eller dokument med blandat innehåll kräver att framför allt grafiska objekt skall kunna fungera självständigt. De olika programmen i en skrivbordsmiljö behöver helt enkelt anropa varandra. Därför har KDE skapat en egen komponentmotor med

namnet DCOP[33] Ovanpå DCOP finns även ett ramverk för att väva in applikationer i varandra (typexemplet är en formeleditor inuti en ordbehandlare) med namnet KParts.[34]

6.6.3 GTK+ och GNOME

Officiell hemsida: `http://www.gnome.org/`

När projektet KDE kommit igång, eller snarast precis samtidigt som det kom igång, uppstod en inflammerad debatt om KDE:s förhållande till det norska Trolltech och dess widget-uppsättning Qt. Flera företrädare för fri mjukvara tyckte att det var oansvarigt att använda Qt, eftersom detta vid tillfället *inte* var fri mjukvara. Källkoden fanns tillgänglig, och Qt kunde laddas ned gratis, men det var förbjudet att sprida modifierade versioner av Qt, eller att tillverka proprietära program för Qt utan att betala licensavgift.

Matthias Ettrich som startat KDE var pragmatiker: även om KDE i sig självt använde licensen GPL, var han inte av åsikten att alla borde eller måste göra det. Han hade snarast målet om ett väl fungerande system för ögonen, inte några krav om att det nödvändigtvis skulle vara helt fritt, i GNU-projektets anda.

Mexikanaren Miguel de Icaza var en av de som tidigt imponerades av KDE och diskuterade därför projektet med såväl Red Hat, som med GNU-projektets ledare Richard Stallman. Icaza ville att Free Software Foundation skulle stödja KDE, men den proprietära licens som användes av Qt gjorde detta omöjligt, eftersom spridningen av sådan mjukvara motverkade FSF:s mål.

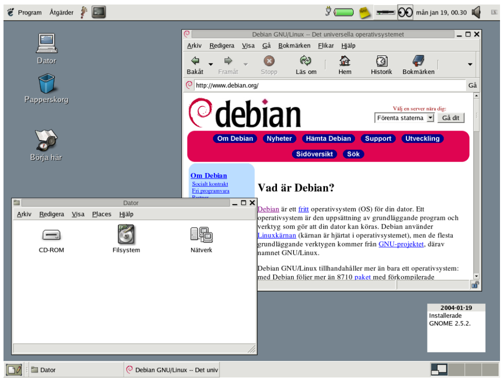
Icaza kontaktade då Trolltech via epost och frågade om de kunde tänka sig att byta licensen för Qt till GNU-projektets GPL. Möjligen var denna kontakt en smula odiplomatisk eftersom han passade på att påpeka en rad felaktigheter som Trolltech skrivit om GPL på sin webbsida. Han fick inga svar från Trolltech.

Efter att ha insett att detta inte skulle lösa problemen försökte han tillsammans med några vänner att arbeta för att förbättra fönsterhanteraren GNUStep, vilket visade sig lönlöst.

I augusti år 1997 bestämde han sig därför att tillsammans med några kamrater skriva *GNOME*. GNOME är, förutom att ordet betyder "tomte", en akronym för *GNU Network Object Model Environment*. Denna

[33]ursprungligen med namnet KOM (för att travestera Microsofts COM (Component Object Model) och baserad på standarden CORBA (Common Object Request Broker Architecture) skapat av Object Management Group. KDE har senare övergett KOM och CORBA, eftersom det inte var tillräckligt väl anpassat för KDE:s behov.

[34]KParts kallades tidigare (under KOM) för OpenParts.



Figur 6.17: Skrivbordsmiljön GNOME har element av Macintosh:s startmeny överst (detta går att välja), en navigationslist nedtill med val av virtuell skrivbordsyta och lista över körande program. En stor mängd program installeras normalt tillsammans med GNOME, ett par av dem syns här.

kryptiska förkortning hade sitt ursprung i att Icaza imponerats av den komponentobjektmodel som utvecklats på Microsoft (omväxlande känd som OLE (Object Linking and Embedding), ActiveX och COM (Component Object Model)). Ett programmeringsprojekt för en liknande struktur för GNU/Linux i grafiska miljöer hade lett fram till att han skrivit en del kod under namnet GNOME och som nu kunde återanvändas.[35]

En av Miguels kamrater var Federico Mena, som arbetat mycket med ritprogrammet The GIMP, en Photoshop-klon för POSIX-system och en del av GNU-projektet. Han kände till att en av författarna till programmet, Peter Mattis, hade tröttnat på att försöka använda wigetuppsättningen Motif för att bygga programmet, och därför skrivit en egen med namnet *GTK+*: *The GIMP Toolkit*. Plustecknet kom sig av, att när det första gången skulle integreras med The GIMP fick det skrivas om till stora delar, och kallades därefter GTK+.

I detta läge beslutade alltså Icaza att bygga en hel skrivbordsmiljö för GNU/Linux, i grunden baserad på GTK+, under namnet GNOME. Senare samma år började Red Hat att stödja projektet med sitt utvecklingslaboratorium, där flera anställda kunde arbeta heltid med att bara utveckla GNOME.

Version 1.0 av GNOME släpptes i mars år 1999. De första versionerna av GNOME var instabila och svåra att använda, men sedan version

[35]Detta är den del av GNOME som numera kallas för *ORBit* (motsvarar KDE:s DCOP) och *Bonobo* (motsvarar KDE:s KParts).

2.0 som kom i juni 2002 har projektet blivit allt mer stabilt och lättrörligt. Det har också vuxit enormt i popularitet visavi KDE och när Sun Microsystems år 2000 annonserade att de avsåg att byta ut CDE mot GNOME i sitt operativsystem Solaris var detta ännu en stor framgång för projektet.

Sedan Trolltech år 1999 släppte Qt under GPL-licensen har den ursprungliga anledningen till att GNOME skapades helt försvunnit, och världen har istället fått två olika skrivbordssystem.

Till viss del har detta naturligtvis varit ett slöseri med resurser, men det är då värt att betänka, att programmen från de två projekten kan samexistera till viss del. T.ex. föredrar många KDE-användare epostprogrammet Ximian Evolution, trots att KMail är det som följer med KDE. Det är inte heller säkert att Qt skulle ha släppts under GPL om inte GNOME uppstått.

GNOME är i dagsläget tänkt att integreras med OpenOffice.org, så att detta blir en naturlig del av GNOME. Som webbläsare används Epiphany, som i sin tur är baserad på Mozilla.

Det är också värt att poängtera, att GNOME är och alltid har varit en del av det större GNU-projektet.

6.7 VNC

Förkortningen VNC skall utläsas *Virtual Network Computing* och springer ursprungligen ur ett projekt på AT&T Cambridge i England. Termen kommer från ett projekt med en ”supertunn” klientdator med namnet *Videotile*. Detta var en dator med en enkel LCD-skärm och en styrpenna som enda inorgan. Datorn körde AT&T:s eget operativsystem ATMos, ett operativsystem som liksom Videotile-datorn själv var utvecklat speciellt för att utnyttja höghastighetsnätverk av typen ATM (Asynchronous Transfer Mode). Tanken var att nätverket skulle kunna vara så snabbt att allt som hände på skärmen och alla inmatningar som gjordes på den skulle kunna skickas över nätverket utan att användaren upplevde någon märkbar fördröjning.

Konceptet är mycket likt det som gällde fönstersystemet *X*, men blev till skillnad från det senare aldrig någon framgång, i varje fall vad beträffar billighetsdatorn Videotile.

Emellertid kom det nya protokoll som användes av Videotile, nämligen VNC, att bli framgångsrikt. Detta utvecklades så att en klient som t.ex. Videotile skulle kunna ansluta till en dator med vilket operativsystem som helst, och ta kontroll över den grafiska användarmiljön.

VNC har i stort mycket gemensamt med *X*. Det har ungefär samma mål och ambitioner: att använda billig hårdvara och erbjuda nätverk-

stransparens. Den udda terminologin från *X* återfinns däremot inte hos VNC: programmet du använder för att ansluta dig med kallas **vncviewer** och är en klient, medan den dator du ansluter till kör en *server*.

VNC:s huvudsakliga användningsområde har blivit *fjärrstyrning*: i synnerhet används det mycket av personer som i sitt arbete behöver kunna kontrollera flera datorer, som ibland är placerade i serverrum och på andra svåråtkomliga ställen. Det används också för att kunna använda andra operativsystem än det egna, genom att öppna en VNC-session mot detta andra operativsystem. Det andra systemets skrivbordsmiljö kan öppnas i form av ett fönster, och behöver inte ta upp hela skärmen på den anropande datorn, även om många använder det så.

Vad som huvudsakligen skiljer VNC från *X* är att VNC inte kan transportera enskilda fönster över nätverket. *X* kan öppna ett *enstaka program* och skicka dess gränssnitt till en annan dator: VNC kan bara skicka över hela den grafiska användarmiljön. När VNC körs under *X* skapas en separat session för detta, så att VNC inte stör det samtidiga arbetet på datorn det ansluter till, men i andra operativsystem kan VNC helt ta över mus och tangentbord, så att maskinen omöjligen kan användas från två platser samtidigt.

VNC finns i dagsläget till bland annat GNU/Linux, Microsoft Windows och MacOS X. I flera distributioner kommer det förinstallerat.

KAPITEL 7

Kringutrustning

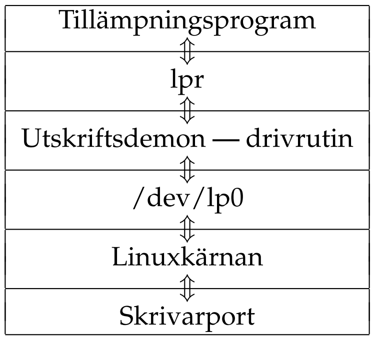
Detta kapitel avhandlar olika former av kringutrustning, d.v.s. saker du ansluter till din dator med olika former av sladdar. Undantaget är mus, tangentbord och bildskärm (som behandlas i kapitlet om fönstersystemet *X*) och modem, som behandlas i kapitlet om nätverket.

Hanteringen av denna typ av utrustning ligger i varierande grad utanför själva Linuxkärnan. Detta är bra, eftersom det ger ett rent gränssnitt mellan olika delkomponenter i systemet och gör kärnan liten och kompakt. Det gör det också möjligt att uppgradera en systemkomponent utan att en annan systemkomponent påverkas av detta.

7.1 Skrivarhantering: CUPS och Foomatic

Alla program som gör utskrifter i GNU/Linux gör det genom att anropa programmet **lpr**, och skicka en PostScript-fil till specialfilen *stdin* kopplad till det programmet. Programmet **lpr** kan också ta råtext, som redan beskrivits, t.ex. **lpr foo.txt**. Om du har en PostScript-fil som du vill skriva ut, kan du helt enkelt skriva **lpr foo.ps**. Alla utskrifter som skickas till **lpr** skickas i sin tur vidare till en demon, som tar hand om utskriften och ser till att den skrivs ut på någon skrivare.

Demonen skriver ut utskriften på en skrivare genom att skicka en ström av binär information till skrivarporten på den dator där skrivaren är ansluten. Vanligtvis är detta samma dator som demonen själv körs på. Om informationen skickas till */dev/lp0* kommer Linuxkärnan i



Figur 7.1: Utskriftsystemet i GNU/Linux så som det kan se ut på en dator med lokal skrivarkö i form av en utskriftsdemon. Här visas det vanligaste fallet — en skrivare kan också vara ansluten till USB-porten.

princip bara att se till att informationen som kommer in i denna teckenenhetsfil går rakt igenom och ut på skrivarporten. Linuxkärnan har alltså i sig ingen information om olika skrivare och deras eventuella drivrutiner, och vet ingenting om hur skrivare egentligen fungerar: det enda den känner till är skrivarportar. Skrivare kan också vara anslutna via USB, vilket behandlas på ett liknande anonymt vis.

All information om hur skrivaren *egentligen fungerar*, och därmed alla drivrutiner för olika slags skrivare, finns i utskriftsdemonen som tar hand om de utskrifter som skickas till den via **lpr**-kommandot. Sammantaget ser systemet ut som i figur 7.1

Skrivardemonen har även hand om en s.k. utskriftskö, som gör att flera program kan skriva ut samtidigt utan att det för den sakens skull blir bråk om skrivarporten på datorn. Skrivardemonerna kan även välja att inte skicka utskriften till den egna datorns skrivare, utan att skicka den till en skrivare på en annan dator, via nätverket.

Skrivardemoner i GNU/Linux hanteras nästan alltid av utskriftssystemet *CUPS*. CUPS är en akronym för *Common UNIX Printing System*, men det borde snarare heta *Common POSIX Printing System*, eftersom det används framför allt av GNU/Linux-varianter.

Före CUPS fanns LPD (*line printer daemon*), LPRng (*line printer next generation*), och en mängd andra utskriftssystem. De är numera väsentligen utdöda — alla använder CUPS, som lyckats etablera en bred standard för hur skrivare skall fungera i alla POSIX-system. CUPS används även av Solaris och MacOS X.

CUPS bygger på IETF:s standard IPP *Internet Printing Protocol*, som beskrivs i en mycket omfattande mängd RFC-dokument. Det är i grund

och botten en demon som körs på en dator med en ansluten skrivare. Demonen tar hand om utskriftsjobb från den egna datorn eller andra datorer i det lokala nätverket, och ser till att de blir utskrivna. Alla utskrifter till CUPS skall som tidigare nämnts skickas i formatet PostScript, som är ett sidbeskrivningsspråk uppfunnet av John Warnock på Adobe i slutet av 1970-talet. På så vis behöver GNU/Linux-program inte bekymra sig om annat än att producera PostScript-kommandon för det de vill skriva ut, resten sköter CUPS.

CUPS-demonen har ett eget användagränssnitt — det är en webbsida på den dator där demonen installerats. Om du arbetar på samma dator som kör CUPS är adressen:

```
http://localhost:631/}
```

Härifrån kan utskriftsjobben övervakas och styras. Om CUPS kör på någon annan dator i nätverket får du naturligtvis lov att ange DNS-adressen eller IP-numret för datorn istället för *localhost*. Det krävs att du loggar in med rootlösenordet för att administrera skrivaren via webbgränssnittet.

CUPS är inte helt homogent: dels använder det i sin tur ESP Ghostscript för att konvertera utskrifterna som kommer i PostScript-format till sitt eget rastergrafikformat, dels använder det drivrutindatabasen Foomatic som ser till att CUPS i slutänden använder rätt drivrutin för att hantera utskriften, och som också tar hand om att formatera utskriften på rätt vis.

De grafiska skrivbordsmiljöerna har byggt egna verktyg som i sin tur använder sig av CUPS: KPrint för KDE och GNOME:s verktyg, som fyndigt nog heter bara *utskriftshanteraren*. För att göra utskrifter från en ensam dator med en skrivare ansluten i den egna skrivarporten[1] brukar dessa verktyg räcka till alldeles utmärkt för att konfigurera och kontrollera utskrifter.

Konfigureringsprogrammen har möjlighet att använda IPP-protokollet för att fråga CUPS-demonen om skrivaren har några speciella inställningar, t.ex. om den kan skriva ut i färg eller dubbelsidigt, eller om den kan häfta samman pappren. Om sådana speciella kommandon stöds, kan programmet som utför utskriften fråga användaren om dessa ”kryddor” önskas,[2] och i så fall haka på denna information i PostScript-filen innan utskriften skickas vidare till **lpr**. **lpr** skickar därefter

[1]Skrivarporten (*printer port*) på en PC är en 25-polig DSUB-kontakt och kallas ibland för LPT1 efter den gamla CP/M- och DOS-namngivningen. Du kan som sagt också ha anslutit en skrivare till USB-porten.

[2]Extrainformationen, ”kryddorna” består i själva verket av en s.k. PPD-fil, PostScript Printer Definition.

hela den resulterande PostScript-filen till skrivardemonen, som tolkar extrainformationen och konverterar PostScript-filen till en rasterbild, och sedan i slutänden skickar denna rasterbild till själva skrivaren.[3] Extrainformationen skickas i slutskedet direkt till skrivarens drivrutin.

Att lägga på sådan extrainformation på utskriften stödjs i varierande grad av olika skrivare och konfigurationsprogram. KDE:s skrivarkonfigurationsprogram KPrint är t.ex. mycket välutvecklat och har stöd för att hantera de flesta sådana "kryddor". Inställningarna kan göras i KDE:s kontrollcenter.

De grafiska gränssnitten kan ibland hantera andra utskriftsystem än CUPS, t.ex. LPRng, Windowsskrivare och Hewlett-Packards JetDirect. Det gör det i och för sig möjligt att till viss del använda andra skrivarsystem än CUPS, men det är sällan detta behövs. I ett rent, modernt GNU/Linux-system med flera datorer och en gemensam skrivare, är en av datorerna kopplad till till skrivaren, och alla andra datorer använder sedan i sin tur skrivaren via nätverket och CUPS.

Med hjälp av programmet Samba kan en CUPS-skrivare ansluten till ett GNU/Linux-system göras tillgänglig även för Windowsanvändare på ett lokalt nätverk, se avsnitt A.6 på sidan 413.

7.2 Scanner

Officiell hemsida: `http://www.sane-project.org/`

Precis som är fallet med skrivare, så ligger all hantering av bildinläsare (engelska: *scanner*, ibland slarvigt kallade "skannrar") helt utanför Linuxkärnan. Program som använder bildinläsare använder SCSI-gränssnitten, USB-portarna eller rent av skrivarporten på datorn, men den enda roll kärnan har är i princip att skicka vidare signalen till den fysiska porten.

Ett fristående och helt dominerande projekt tar hand om uppgiften att hantera bildinläsare: programbiblioteket *Sane* (Scanner Access Now Easy). Detta fyller exakt samma funktion som scannersystemet TWAIN[4] som används i Macintosh- och Microsoft Windows-system.

Du kan testa om Sane kan hitta din scanner genom att köra kommandot **sane-find-scanner** som följer med programbiblioteket. Om detta program kan hitta scannern, kommer alla andra program som använder Sane förmodligen också att göra det.

[3] Om skrivaren kan tolka PostScript direkt kommer sidorna naturligtvis inte att rastreras.

[4] Namnet TWAIN kommer från en idiomatisk fras av Rudyard Kipling: *…never the twain shall meet…* men många läser ändå ut det som en akronym för *"Technology Without an Interesting Name"*.

Tillämpningsprogram
⇕
Xsane, Kooka, etc.
⇕
Programbiblioteket Sane
⇕
/dev/sg0, /dev/lp0, /dev/usb/scanner0 ...
⇕
Linuxkärnan
⇕
SCSI-port, Skrivarport, USB-port

Figur 7.2: Scannersystemet i GNU/Linux, översiktligt.

Till detta bibliotek finns de grafiska användargränssnitten *Xsane*[5] (*X*:et kommer naturligtvis av att programmet är skrivet för fönstersystemet *X*) som är baserat på GTK+ och *Kooka*[6] som används av KDE. Det senare kan även utnyttja GNU-projektets *Ocrad* eller det fristående *GOCR* som är program för att tolka textmassor från inlästa dokument, s.k. textigenkänning (på engelska kallat för *OCR*, *Optical Character Recognition*).

Dessa användargränssnitt kan användas fristående för att läsa in bilder, men det vanligaste är att de i sin tur startas av andra program som t.ex. The GIMP,[7] där inlästa bilder kan beskäras och bearbetas.

Såväl programbiblioteket Sane som ett eller flera grafiska användarverktyg följer med i de flesta GNU/Linux-distributioner. Om du vill använda en scanner ansluten till den egna datorn är det oftast bara att starta Xsane eller Kooka (eventuellt från ett annat program som The GIMP) och läsa in bilden.

Om du vill ha en scanner ansluten till en enda dator i ett nätverk och låta andra datorer ansluta sig till denna dator för att utföra bildinläsningar finns en scannerdemon med namnet **saned** som medföljer programbiblioteket Sane. Denna demon startas vid behov från **inetd** eller **xinetd** (se avsnitt 9.4 på sidan 329) och brukar vara deaktiverad i de flesta distributioner. Manualsidan för **saned** ger mer information för den som behöver använda detta system.

[5]Se: `http://www.xsane.org/`
[6]Se: `http://www.kde.org/apps/kooka/`
[7]Program som The GIMP avhandlas i kapitel 11 om tillämpningsprogram.

7.3 USB-enheter

USB-enheter är all form av modern utrustning som ansluts till USB-portarna på din dator. Moderna IBM PC-kompatibla datorer har ofta minst två USB-portar, och dessa kan sedan grenas ut med s.k. USB-hubbar så att upp till 127 olika USB-enheter (teoretiskt sett) kan anslutas till en och samma dator. USB är en de facto-standard definierad av USB Implementers Forum och ersätter successivt alla gamla anslutningsdon för modem, skrivare, scanner, externa hårddiskar etc. i syfte att skapa ett mer enhetligt och intelligent system för externa enheter. Även tangentbord, högtalare och musar ansluts nu allt oftare via USB.

Ett av huvudmålen med USB-specifikationen är att externa enheter inte skall fordra speciella drivrutiner, utan tillhöra en s.k. *enhetsklass* som anger hur enheten skall bete sig visavi operativsystemet, oavsett vilket operativsystem detta nu råkar vara. På så vis blir det allt lättare att använda ny kringutrustning med GNU/Linux.

USB-systemet i GNU/Linux bygger på två separata delar:

- *Kärnans drivrutin för den fysiska USB-porten.* Denna kan vara en av tre typer: OHCI (Open Host Controller Interface), UHCI (Universal Host Controller Interface) eller EHCI (Extended Host Controller Interface).

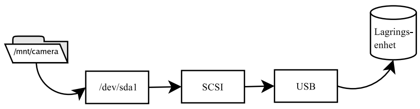
  OHCI och UHCI är drivrutiner för USB 1.1-specifikationen, medan EHCI används för USB 2.0. På nyare datorer är det således den senare som skall användas. Stödet för EHCI i 2.4-versionen av Linuxkärnan är dessvärre ganska opålitligt, och det kan därför vara nödvändigt att istället använda någon av de gamla drivrutinerna. I 2.6-versionen av kärnan skall EHCI fungera som tänkt.

- *Hotplug-systemet.* USB-enheter är gjorda för att kunna kopplas in och ut under drift. Kärnan kommer då att lägga till, respektive ta bort gränssnittet ur kärnan, och anropar sedan programmet **/sbin/hotplug**.[8] Detta program delegerar i sin tur arbetet till skriptet */etc/hotplug/usb.agent* som i sin tur kan använda ytterligare skript på samma plats.

  Om några speciella moduler behöver laddas in i kärnan för att hantera den enhet som just anslutits, kommer skripten i */etc/hotplug* att se till att dessa moduler laddas. Om modulerna inte finns tillgängliga kan det bli problem.

  Skripten i */etc/hotplug* kommer även att anropas när utrustningen kopplas loss.

[8]Hotplug används även för andra enheter som kan kopplas in under körning, t.ex. firewire (IEEE 1394), vissa nätverkskort, vissa PCI-enheter o.s.v.



Figur 7.3: Schematisk bild av hur Linuxkärnan använder USB-masslagringsenheter. I detta exempel har en digitalkamera monterats som */mnt/camera*.

7.3.1 USB-lagringsenheter, digitalkameror etc.

USB kan användas för att ansluta såväl mus som tangentbord och t.o.m. högtalare till datorn, men den vanligaste typen av USB-enheter är antagligen s.k. ”USB mass storage”-enheter. Denna enhetstyp används av USB-nycklar (den typ av USB-enheter som många har som nyckelring och universell diskett) och av många digitalkameror, MP3-spelare och portabla hårddiskar.

Varje USB-enhet som ansluts till en buss tillhör en s.k. enhetsklass. Den klass som gäller för ”USB mass storage” är gemensam för alla enheter av denna typ. Denna klass kommer alltid att hanteras på samma vis av hotplug-systemet, och din Linuxkärna behöver alltså inte känna till just din typ av enhet för att veta vad som skall göras då den ansluts, det räcker att den avslöjar att den tillhör klassen ”USB mass storage”.

En enhet som använder ”USB mass storage” blir först en USB-enhetsfil någonstans under hierarkin */dev/usb*. Därefter kommer *hotplug*-skripten med hjälp av **modprobe** att ladda en SCSI-lagringsenhetsmodul med namnet *sd_mod* (”storage device”), som i sin tur använder den generella *scsi_mod*. Dessa SCSI-moduler behövs, eftersom dessa lagringsenheter använder en speciell form av SCSI-protokollet över USB-kabeln.

Därefter laddas modulen *usb-storage* in i kärnan, och ansluts mellan USB-enhetsfilen och SCSI-modulen. USB-enheten kommer då att dyka upp som en ren SCSI-enhet med ett namn som t.ex. */dev/sda*. Enheter av det här slaget har sällan mer än en partition. Alla de moduler som nämnts måste finnas tillgängliga i Linuxkärnan eller i form av inladdningsbara moduler, annars kommer det hela att misslyckas, och du kan inte använda din lagringsenhet. De flesta distributioner (men inte alla) kommer färdiga med alla dessa moduler tillgängliga någonstans i */lib/modules*.

USB-lagringsenheten är vanligen sedan partitionerad i olika partitioner som vilken hårddisk som helst, t.ex. kommer antagligen */dev/sda1* att beteckna den första partitionen på lagringsenheten */dev/sda*.

Om denna inte monteras automatiskt kan du vanligen montera den som root med kommandot **mount /dev/sda1 /mnt/foo** förutsatt att mon-

teringspunkten (katalogen) */mnt/foo* existerar.

En förformaterad USB-enhet brukar använda någon variant av filsystemet FAT[9] och detta måste då naturligtvis stödjas av kärnan. Filsystemet FAT är bland GNU/Linux-användare även känt som VFAT, men VFAT betecknar egentligen bara metoden för att lagra långa filnamn i ett FAT-filsystem.[10] Det finns inget krav på att just FAT skall användas på en USB-enhet, men i t.ex. en digitalkamera så skall ju själva kameraoperativsystemet också kunna använda filsystemet, och dessa stödjer oftast bara FAT. Filsystemet på en USB-nyckel kan du byta hur du vill, och du kan även partitionera om nyckeln, men om du vill kunna använda nyckeln tillsammans med Microsoft Windows bör du hålla dig till en partition med FAT-filsystem. De flesta USB-nycklar kommer dessutom förformaterade med detta filsystem. Vissa portabla hårddiskar levereras med filsystemet NTFS förinstallerat, och om din Linuxkärna inte stödjer detta kan du utan vidare formatera om det till t.ex. Ext2.

De flesta distributioner har kommit långt med stöd för denna typ av enheter, och för en vanlig användare kan det ofta räcka med att koppla in USB-enheten för att den skall dyka upp som en ikon på användarens skrivbord.

7.4 CD- och DVD-bränning

Det finns olika slags CD- och DVD-skivor: CD-ROM och DVD-ROM är skivor som tillverkats på fabrik genom en litografisk process. CD-R eller DVD±R är skivor som levereras tomma och kan brännas hemma, en gång. Att skivorna kan brännas *en gång* betyder i detta fall att samma område på skivan bara kan skrivas *en* gång. Du kan sedan i och för sig skriva en liten bit av skivan åt gången, genom att använda olika s.k. *sessioner*. Beteckningen *DVD±R* kommer ur förhållandet att det finns två konkurrerande DVD-standarder för brännbara skivor: en med namnet *DVD-R* och en med namnet *DVD+R*. De flesta moderna brännare och läsare kan hantera båda dessa standarder och använder därför i allmänhet samlingsbeteckningen *DVD±R*. De tomskivor som du kan köpa i handeln brukar i allmänhet vara DVD+R.

CD-RW och DVD±RW är samma sak som CR-R och DVD±R med tillägget att skivan också kan raderas och på så vis användas flera gånger.

[9] *File Allocation Table* finns i varianterna FAT12, FAT16 och FAT32

[10] Tekninken innebär att ett filnamn lagras som upp till extra fyra individuella FAT-katalogposter som har typen ”hidden” och som därmed inte syns, men som då används som förvaringsutrymme för de extra filnamnstecknen.

I princip all form av CD- och DVD-bränning i GNU/Linux utförs med ett programpaket som heter *CDRtools*. Detta är en samling av en rad program. Det huvudsakliga programmet är **cdrecord**, men också t.ex. **mkisofs** som används för att skapa själva filsystemet för en CD- eller DVD-skiva och **cdda2wav** som kan kopiera spår från vanliga ljud-CD-skivor ingår.

Programmen i detta paket är skrivna av Jörg Schilling och används även av BSD-projektet.

Ytterligare ett par viktiga projekt finns: **cdrdao**, som används för att spela in skivor i *disc-at-once*-läget, och **cdparanoia** som är ett avancerat program för att kopiera ljudspår från ljud-CD-skivor till hårddisk. Alla dessa program brukar följa med i de flesta distributioner, och **cdparanoia** är betydligt bättre än **cdda2wav** som medföljer CDRtools.

För DVD-bränning krävs en hel del specialparametrar till **cdrecord** och **mkisofs** och därför används programmen oftast indirekt genom programmet **growisofs**. Detta och ett antal andra verktyg, bland annat **dvd+rw-format** som används för att formatera DVD-skivor som både är läs- och skrivbara ingår i paketet *DVD+RW-tools*.

Linuxkärnan har i sig inget speciellt stöd för att bränna CD-skivor: detta sköts istället helt av **cdrecord** eller **cdrdao**. I Linuxkärnan version 2.4 bygger all kommunikation med brännaren på att kommandon skickas via SCSI-gränssnittet. Detta innebär att brännaren alltid måste vara installerad så att den betraktas som en SCSI-enhet. Detta är inte nödvändigt i version 2.6 av kärnan.

Många brännare är idag inte av SCSI-typ utan IDE-typ,[11] och i sådana fall måste linuxkärnan i version 2.4 konfigureras om så att IDE-CD-brännare används genom att emulera SCSI, istället för att använda IDE direkt. Detta är inte speciellt enkelt och är därför ett problem för många användare. (Om du istället har skaffat en CD-brännare av SCSI-typ är detta naturligtvis inget problem.) I den nya 2.6-kärnan är detta problem åtgärdat, och det skall vara möjligt att bränna CD-skivor utan att använda SCSI-emulering. Bortse i så fall från följande utvikning.

För att slå på SCSI-emulering i kärnan finns två vägar:

- Om du har såväl ”SCSI support”, ”SCSI CD-ROM support” ”IDE/ ATAPI CDROM support” och ”SCSI emulation support” installerat i form av moduler, kan du ge ett kommando till kärnan under uppstart (genom att modifiera uppstartfilerna för LILO eller GRUB, se sidan 184), som ser till så att IDE CD-brännaren förvandlas till en SCSI CD-brännare. Heter din IDE CD-läsare t.ex. */dev/hdc* är kommandot för LILO, i din *lilo.conf*-fil:

[11] IDE (Integrated Drive Electronics) är som tidigare nämnts ett annat namn för ATA (Advanced Technology Attachment).

```
append="hdc=ide-scsi"
```

I GRUB är det bara att lägga till detta i slutet av en rad som specificerar önskad kärna, t.ex.:

```
kernel /vmlinuz-2.4.20 ro root=/dev/hda2 hdc=ide-scsi
```

Utöver detta måste du också instruera din kärna att ladda in SCSI-emuleringsmodulen med **modprobe ide-scsi** eller liknande kommando, i systemet uppstartfiler.

Efter detta kommer din CD-brännare istället (i just detta fall) att dyka upp som blockenheten */dev/shc*. Lokala variationer förekommer.

- Om du vill styra om din IDE CD-brännare ”hårt” t.ex. i en kärna utan moduler, måste ”SCSI support”, ”SCSI generic support” och ”SCSI CD-ROM support” slås på, medan ”IDE/ATAPI CDROM support” skall slås av, och ”SCSI emulation support” slås på. Kärnan måste sedan kompileras om och installeras. Detta medför att enhetsfilen för CD-spelaren, som tidigare kanske var en IDE-enhet som hette */dev/hdc* förvandlas till */dev/scd0*. Den kan fortfarande monteras och användas som vanligt, men numera bara som om den vore en SCSI-enhet.

Det finns också en tredje typ av CD-brännare som använder parallellporten. Dessa är om möjligt ännu svårare att få att fungera (även under Microsoft Windows!), och kommer inte att behandlas här. Istället hänvisas till CD-Writing HOWTO[12] (som även innehåller en hel del annan matnyttig information.)

I det följande kommer att beskrivas hur en CD-skiva kan skapas med hjälp av främst kommandoradsverktygen **mkisofs** och **cdrecord**.

Flera grafiska program har skapats för att förenkla användandet av de helt kommandoradsbaserade verktyg som ligger till grund för CD-bränningssystemet i Linux, t.ex. *gtoaster* för GNOME eller *K3b* för KDE. Prova dem gärna när du väl fått din brännare att fungera!

7.4.1 Skapa en Data-CD/DVD

Att ”för hand” (d.v.s. utan grafiska hjälpverktyg) skapa en CD- eller DVD-skiva kräver två steg:

[12]Se: `http://www.tldp.org/HOWTO/CD-Writing-HOWTO.html`

- Skapa först ett filsystem i form av en stor fil, (en s.k. ISO-fil) som innehåller allt du vill lägga på CD-/DVD-skivan.
- Bränn den stora filen till en tom CD-/DVD-skiva.

Skapa ISO-filen

De stora filerna som i sin tur innehåller filerna du vill bränna kallas ISO-filer eftersom de innehållet ett filsystem av en typ som definieras i standarddokumentet ISO 9660. De flesta skapar de stora ISO-filerna med kommandot **mkisofs**, som kan användas på följande vis:

```
# mkisofs -r -iso-level 2 -o foo.iso /bar
```

Detta tar alla filer och kataloger under katalogen */bar* och kopierar in dem i ett ISO-filsystem lagrat i filen *foo.iso*. (Du kan naturligtvis inte lägga filen du skapar i samma katalog som filerna du kopierar, du kan nog själv räkna ut vad som händer då.) Flaggan **-r** talar om att ägarskap och skrivbarhet av filerna skall avlägsnas (detta är oftast inte meningsfullt på en CD-skiva), **-iso-level 2** gör det möjligt att åtminstone ha långa filnamn (**-iso-level 3** passar bättre för svenska men kanske inte alltid kan användas av systemet du skall läsa skivan på) och flaggan **-o foo.iso** anger att filen som skapas skall heta så.

En ISO-fil är i praktiken bara en lång rad bytes, med ett filsystem definierat enligt ISO 9660. Anledningen till att detta system används är att det stödjs av så gott som alla operativsystem, t.ex. Microsoft Windows eller MacOS X. **mkisofs** följer inte slaviskt ISO 9660, utan använder utökningarna JOLIET (långa filnamn i Windows), HFS (Macintosh långa filnamn) och Rock Ridge Extensions (POSIX-filinformation) för att lagra filerna i filsystemet.

Om du vill kan du montera din ISO-fil innan du bränner den. Detta kan också användas för att titta på innehållet i andra ISO-filer som du får tag på:

```
# mount -t iso9660 -o loop foo.iso /mnt/foo
```

Detta kommer att montera ISO-filen *foo.iso* i monteringspunkten */mnt/foo*, förutsatt att det finns en katalog med det namnet. (Monteringen använder det s.k. loopback-gränssnittet, som beskrevs i avsnittet om krypterade hårddiskar i säkerhetskapitlet på sidan 353.) Montera av filen igen med **umount /mnt/foo**.

Om du vill tillverka en ISO-fil av en CD-skiva du redan har kan du skriva:

```
# dd if=/dev/cdrom of=foo.iso
```

Detta förutsatt att din CD-läsare är kopplad till enhetsfilen */dev/cdrom*. (Om din CD-läsare är t.ex */dev/hdc* kan du helt enkelt göra en symbolisk länk från *hdc* till *cdrom* för att lösa problemet.) Är det någon form av fel på CD-skivan kommer operationen att misslyckas: då kan du be kommandot **dd** att fortsätta trots felen med parametern **noerror**, men många tycker kanske att det är det bättre att montera skivan och kopiera över alla hela filer, en fil i taget:

```
# mount /mnt/cdrom
# mkisofs -r -o foo.iso /mnt/cdrom
```

Det finns egentligen inget som helst krav på att en ISO-fil skall innehålla ett ISO 9660-filsystem (förutom att namnet så att säga antyder det). När filen bränns betraktas den ändå bara som en lång rad bytes lagda efter varandra. Du kan skapa en skiva med vilket filsystem som helst på, t.ex. Ext2:

```
# dd if=/dev/zero of=foo.iso bs=1M count=650
# mkfs -t ext2 foo.iso
# mount -t ext2 -o loop foo.iso /mnt/foo
```

Detta skapar en ”ISO-fil” på 650 megabyte (vilket ju är den storlek som ryms på en vanlig CD-skiva) som innehåller ett Ext2-filsystem. (**mkfs** kommer säkert att varna om att du försöker skapa ett filsystem inuti en fil istället för en blockenhet, men det behöver du inte bry dig om.) Den monteras sedan via ett loopback-gränssnitt och kan därefter fyllas med innehåll genom att du kopierar filer till */mnt/foo* tills filen blir full. Då avmonterar du filen och bränner den. Det går naturligtvis bra att kryptera innehållet också genom att använda en krypterad loop.

CD-skivor som inte innehåller ett ISO 9660-filsystem kommer oftast inte att monteras automatiskt under GNU/Linux, eftersom */etc/fstab* vanligen anger att enheten */dev/cdrom* skall monteras som ett iso9660-filsystem. Du får därför montera skivan för hand, eller ändra i */etc/fstab* (filsystemstypen **auto** kommer t.ex. antagligen att detektera om CD-skivan innehåller Ext2).

Om du skall använda CD-skivan med något annat operativsystem än GNU/Linux eller t.ex. i en MP3-spelande stereo eller DVD-spelare till din TV bör du hålla dig till ISO-formatet.

DVD-skivor kan innehålla större filsystem än CD-skivor. En vanlig DVD±R kan innehålla hela 4,7 gigabyte![13]

Bränn ISO-filen

För att bränna en CD-skiva skall du utnyttja **cdrecord** eller **cdrdao**. Dessa kommandon måste i princip *alltid* köras som root. Det första du måste veta är vad din brännare har för SCSI-ID. Du får en lista över tillgängliga SCSI-enheter med kommandot:

```
# cdrecord -scanbus
```

En av dessa måste vara din CD-brännare, annars har du inte lyckats installera SCSI-emuleringen ordentligt i din dator. (Gå i så fall tillbaka och ordna det först.) SCSI-ID:t har oftast formen 0,N,0 där *N* är en siffra mellan 0 och 9, t.ex. 0,0,0 eller 0,5,0. När du väl vet vilket SCSI-ID din CD- eller DVD-brännare har är det bara att bränna:

```
# cdrecord -v speed=2 dev=0,N,0 foo.iso
```

Här skall **speed** sättas till den hastighet du vill använda (mindre än eller lika med din brännares topphastighet) och **dev** skall sättas till det SCSI-ID som du just tagit reda på. Dessa båda parametrar måste *alltid* anges.

Om du har en 2.6-kärna och vill använda din IDE-baserade CD-brännare utan SCSI-emulering behöver du inte veta någonting om SCSI-ID eller liknande. Om din brännare heter t.ex. */dev/hdc* skriver du bara:

```
# cdrecord -v speed=2 dev=/dev/hdc foo.iso
```

Motsvarande kommando för en DVD-skiva är:

```
# growisofs -Z /dev/hdc=foo.iso
```

För att radera CR-RW-skivor innan de skrivs kan du ange växeln **blank=fast** (utan minustecken framför) till **cdrecord**. Detta raderar skivan först, och skriver sedan det nya innehållet. Parametern **-Z** till **growisofs** har samma effekt för DVD±RW-skivor. Om du vill skapa en DVD-skiva som innehåller en film som skall kunna spelas i en vanlig DVD-spelare bör du också haka på parametern **-dvd-compat** till **growisofs**.

[13] De tomskivor som finns i handeln när detta skrivs har 2295104 sektorer à 2048 bytes vardera, och en ISO-fil som precis fyller hela skivan kan därför skapas med t.ex.: **dd if=/dev/zero of=$1 bs=2048 count=2295104**

DVD-brännare kräver som du ser inga hastighetsparametrar: om inget anges kommer den högsta möjliga hastigheten att användas.

Om du måste så *kan* du med hjälp av **mkisofs** och **cdrecord** skapa en CD-skiva och bränna den samtidigt, utan att först göra en ISO-fil. Du kan skriva t.ex.:

```
# size=`mkisofs -print-size -quiet /foo`
# mkisofs /foo | cdrecord -v tsize=$size speed=2 dev=0,N,0
```

För DVD-skivor används istället genvägen med **growisofs** som är gjort för att förenkla operationer av detta slag:

```
# growisofs -Z /dev/dvd -R -J /foo
```

Det finns en risk med detta: kanske hinner inte datorn med att göra båda sakerna samtidigt. I detta fall går bränningen åt skogen och du har förstört en CD-skiva. Å andra sidan kan det ju hända att du har ont om diskutrymme, och då är detta enda utvägen.

7.4.2 Skapa en Audio-CD

Att skapa en Audio-CD kräver att du har tillgång till ljudet som skall brännas i form av enkla s.k. CDR-filer (CD-record), eller ”WAV-filer”[14] som innehåller ljuddata i PCM-format[15] inspelade i 44100 Hz, 16-bit stereo. Detta är nämligen det format som CD-skivor använder.

”WAV-filer” är egentligen bara helt vanlig 44100 Hz 16-bit stereo PCM-data med några extra byte (en RIFF-header) i början av filen, och CDR-formatet är samma sak minus de extra byten. Vilken sort som används spelar ingen större roll, men ”WAV-filer” är vanligast.

Dessa CDR- eller ”WAV-filer” kan åstadkommas på olika vis:

- Om du redan har tillgång till filerna i ett annat format, t.ex. MP3 eller Ogg Vorbis kan du antagligen konvertera dem till CDR-filer eller ”WAV-filer” med någon mediaspelare. XMMS kan t.ex. konfigureras att skriva utdatan från programmet till filer på hårddisken, vilket allt som oftast är 44100 Hz 16-bit stereo ”WAV-filer”. Även enkla kommandoradsprogram som t.ex. **mpg321** kan åstadkomma detta:

[14]”WAV-filer” är ett farligt begrepp: filer från Microsoft Windows med filändelsen *.wav* betecknar en slags containerfil (RIFF, Resource Interchange File Format) och kan i princip innehålla vad som helst. När jag skriver ”WAV-fil” är detta för att beteckna PCM-rådatafiler.

[15]PCM-format (Pulse Coded Modulation) innebär i detta fall att ljudtrycket i varje tidsögonblick med intervallet $\frac{1}{44100}$ sekund omvandlas till ett numeriskt värde mellan 0 och $2^{16} - 1$.

```
# mpg321 --wav foo.wav foo.mp3
```

- Om du bara har tillgång till filerna i form av en ljud-CD-skiva måste skivan först läsas av och omvandlas till ”WAV-filer”. Detta kan göras med programmet **cdda2wav**, men programmet **cdparanoia**[16] rekommenderas framför **cdda2wav**, eftersom detta gör mer ordentliga ansträngningar att kopiera ljudet om det t.ex. skulle vara repor på skivan. För att läsa av en CD-skiva med **cdparanoia** skapar du en tom katalog, där du ”ställer dig” och skriver:

  ```
  # cdparanoia -B
  ```

  Detta kopierar alla spår på musikskivan till ”WAV-filer”. Detta kommer även att kopiera spår som är ”data” från skivor med blandat innehåll, så kontrollera att det du får med verkligen är ljudspår. De flesta CD-skivor brukar även innehålla ett mycket kort ”spår noll” i början av skivan.

- Om du skulle råka ha en fil i något annorlunda format, t.ex. Macintosh AIFF eller 8-bitars rå samplingsdata, kan detta konverteras till 44100 Hz 16-bitar stereo-PCM-filer med t.ex. programmet **sox**.[17]

För att bränna de CDR-filer eller ”WAV-filer” du nu åstadkommit till en tom CD-skiva som skall kunna spelas i en vanlig hemmastereo skriver du:

```
# cdrecord -v -audio speed=2 dev=0,N,0 *.wav
```

Globbningsuttrycket (**.wav*) i slutet av raden kommer dock att välja alla filerna i bokstavsordning, så vill du ha dem i en speciell ordning får du rada upp dem.

Du kan också spela in CD-spåren ett i taget genom att ange flaggorna **-nofix -pad** för varje ”WAV-fil” du bränner, och avsluta med **cdrecord -fix**. Detta lämnar skivan ”öppen” mellan inspelningen av spåren och stänger den sedan med ett speciellt kommando.

Att spela in ljuskivor direkt med **cdrecord** på detta vis skapar ett kort tomrum mellan varje spår på skivan. För det mesta spelar detta ingen roll, men om du vill ha full kontroll över dina ljudskivor, och om

[16] Se: `http://www.xiph.org/paranoia/`
[17] Se: `http://sox.sourceforge.net/`

du vill kunna spela in musik utan avbrott, måste du använda den s.k. *disc at once*-tekniken, vilket innebär att hela skivan bränns i ett svep utan att lasern släcks förrän all data är bränd. Detta stödjs inte av **cdrecord** så i detta fall måste programmet **cdrdao** användas istället. Detta program kräver dock en speciell skriptfil för att kunna göra skivan.

KAPITEL 8

Kompilera själv

Om något skall göras ordentligt: gör det själv!

Den korta historien är som följer. Förutsatt att du laddat ner ett program du vill kompilera i form av ett filarkiv, komprimerat med GNU tar och GNU Zip (vilket är det vanligaste), och förutsatt att programmet följer konventionerna för GNU Autotools (vilket är vanligt) och förutsatt att inga fel inträffar, kommer följande kommandosekvens att kompilera och installera ett nytt program:

```
tar xvfz foo-1.1.0.tar.gz
cd foo-1.1.0
./configure
make
su
make install
```

Detta gör att du packar upp källkoden (rad 1), går in i den uppackade katalogen (rad 2) konfigurerar den med skriptet **configure** (rad 3),[1] kompilerar den (rad 4), växlar till root-användaren (rad 5) och installerar programmet i */usr/local* (rad 6).

Lokala variationer på komprimerade filer förekommer: ibland dras filsuffixet *.tar.gz* samman till *.tgz*. Det är samma sak, och packas upp

[1]Om du inte minns varför du måste skriva **./** före skriptets namn, så gå tillbaka och läs på sidan 58.

precis som ovan. Det är också alltmer populärt att använda komprimeringsprogrammet **bzip2** istället för **gzip**. I sådana fall heter källkodsarkivet **foo-1.1.0.tar.bz2** istället, och packas då istället upp med **tar xvfj foo-1.1.0.tar.bz2**.[2]

Fungerade det inte? Tråkigt. Då är det tyvärr så, att det som följer i detta kapitel inte ger någon fullständig information om vad som krävs för att lösa alla fel som kan tänkas ha uppstått. De vanligaste och mest triviala felen kommer att presenteras, tillsammans med de sammanhang som de hör hemma i. Men för att kunna kompilera alla program som finns fordras det faktiskt att du är mer eller mindre programmerare.

Detta är anledningen till att en "normal" användare bör hålla sig till att använda de färdiga paket som kommer med distributionen, eller paket som är avsedda att användas tillsammans med en viss distribution.[3]

8.1 Kompileringsverktyg

För att kunna kompilera program under GNU/Linux behöver du naturligtvis först och främst tillgång till de verktyg som en programmerare använder. De viktigaste är:

- GNU Compiler Collection (GCC)
- GNU Make
- GNU C Library (glibc), samt eventuellt utvecklarpaket. Ibland kan detta vara uppdelat i flera underpaket, som *glibc-common*, *glibc-kernheaders* o.s.v. Normalt är alla dessa paket installerade med distributionen.
- Standard C++ library (libstdc++), samt eventuellt utvecklarpaket. Utvecklarpaketet till C++ saknas ofta i en standardinstallation och måste installeras separat.

Dessa verktyg kan så gott som alltid installeras direkt från paket som följer med din distribution. (Vad som menas med ett *utvecklarpaket* förklaras i nästa avsnitt.)

Härnäst kommer du att stöta på *beroenden* mellan ditt program och andra systemkomponenter.

[2]Jag ger här bara olika flaggor till kommandot **tar**, flaggan **z** signalerar att det är fråga om ett GNU zip-paket, medan **j** signalerar att det är fråga om **bzip2**. Det går såklart lika bra att först dekomprimera arkiven med **gunzip** eller **bunzip2**, och därefter packa upp dem med **tar xvf foo ...**.

[3]Se även avsnitt 4.6 på sidan 166 om program utanför distributionen.

8.2 Beroenden

För att kompilera ett program måste vissa *beroenden* vara uppfyllda. De vanligaste är:

- Beroenden av vissa verktyg som behövs under kompileringen, främst de som nämndes i föregående avsnitt.
- Beroenden av programbibliotek som tillhandahåller delar av funktionalitet.
- Beroenden av andra installerade program eller vissa konfigurationsfiler.

Den vanligaste typen av beroenden som uppstår när kompileringsverktyg är installerade och kompilering väl kan komma till stånd är *beroenden av andra programbibliotek*.[4] Denna typ av beroenden brukar ge sig tillkänna under konfigureringsfasen. Konfigureringsprogrammet brukar testa två saker: först att en s.k. **.h**-fil finns tillgänglig. Detta betyder att det letar efter en fil med ett gränssnitt till programbiblioteket, och i programspråken C och C++ heter dessa något i stil med *foo.h*.

I den filhierarki som råder i GNU/Linux-system finns dessa filer normalt i */usr/include* eller */usr/local/include* och konfigurationsskripet med namnet **configure** kommer automatiskt att leta efter dem där. Om dina programbibliotek ligger någon helt annanstans, blir du normalt sett tvungen att ange detta för konfigurationsprogrammet med hjälp av någon parameter. Om du inte är herre över systemet du använder kan det t.ex. hända att du har lagt filerna i din hemkatalog,[5] och då blir du tvungen att ange detta till **configure**, oftast i form av en fullständig sökväg till din hemkatalog.[6]

Efter att konfigureringsskriptet letat efter **.h**-filen kommer det normalt också att leta efter själva programbiblioteket, vilket vanligtvis ligger i */usr/lib* eller */usr/local/lib*. Har väl **.h**-filen identifierats brukar inte detta vara något problem.

Ett vanligt problem är att **.h**-filerna inte kan hittas i */usr/include*, fastän själva programbiblioteket finns i */usr/lib*. Detta beror på att många distributioner helt sonika plockar bort **.h**-filerna, eftersom de tar plats och bara används vid kompilering. Själva programbiblioteken behövs ju

[4] Vad programbibliotek eller dynamiska länkbibliotek är, och hur de används i Linuxkärnan förklaras utförligt i avsnitt 5.4.5 på sidan 195.

[5] En del har en hel standardhierarki i sin hemkatalog med */home/foo/lib*, */home/foo/include*, */home/foo/bin* o.s.v.

[6] Du kan normalt använda miljövariabeln **$HOME** som fullständig sökväg till din hemkatalog, undersök gärna denna med **echo $HOME**.

under körning och följer därför alltid med, om de används. Eftersom s.k. ”vanliga” användare inte skall behöva kompilera program, har dessa **.h**-filer därför lagts i ett speciellt paket, oftast med namn med suffix som **-dev** eller **-devel**.

Om ditt program t.ex. använder programbiblioteket *libfoo* betyder detta att ett länkbart programbibliotek skall finnas installerat i form av */usr/lib/libfoo.so.1* (eller liknande) och en **.h**-fil skall finnas i form av */usr/include/foo.h* eller kanske */usr/include/libfoo.h*. Den senare filen kan alltså ha hamnat i ett speciellt *utvecklarpaket*, och detta medför att konfigureringen spårar ur med ett felmeddelande. I distributionen Debian GNU/Linux åtgärdar du i så fall detta med **apt-get install libfoo-dev** och i RPM-baserade distributioner måste du hitta ett RPM-paket med ett namn i stil med **libfoo-devel-1.0.0.i386.rpm**, och installera det. Rör det sig om ett standardbibliotek finns detta paket oftast på skivorna som medföljer distributionen.

I många fall kan ditt pakethanteringssystem hjälpa till att undersöka vilka programbibliotekspaket som finns installerade och vilka de motsvarande utvecklarpaketen är.

När väl alla utvecklarpaket kommit på plats brukar kompileringen gå igenom utan problem. I något fall kan det hända att din version av det önskade programbiblioteket är för gammalt, och du måste i så fall uppgradera biblioteket. Om det rör sig om en standardkomponent i en distribution kan det vara farligt att uppgradera programbibliotek, eftersom dessa i sin tur kan användas av hundratals andra program, så tänk dig för. De som utvecklar programbibliotek brukar i och för sig försöka undvika att orsaka situationer där nyare bibliotek är inkompatibla med gamla, men sådant händer trots allt.

Om programbiblioteket *inte finns alls* på din dator, kan du vara tvungen att först ladda ner och kompilera själva programbiblioteket. Programbibliotek kan sedan i sin tur vara beroende av andra programbibliotek. Det kan bli ganska många olika bibliotek som måste laddas ned och kompileras separat om du har otur.

Ett otrevligt misstag är att ladda ner och kompilera ett programbibliotek trots att detta redan finns tillgängligt, men utvecklarpaketet med **.h**-filerna saknas. I sådana fall kommer **.h**-filen i */usr/local/include* (som du installerar) att användas tillsammans med själva programbiblioteket från */usr/lib*, vilket har företräde framför det nya programbibliotek som du precis installerat i */usr/local/lib*. Om detta rör sig om två helt olika versioner av programbiblioteket kan beteendet bli helt oförutsägbart.

När du väl har installerat ett programbibliotek gäller det att ditt program kan hitta det också. Under kompileringen går detta för det mesta bra, men när programmet skall köras kan ytterligare problem

uppstå: när ett program körs kommer nämligen bara vissa specifika platser att genomsökas efter programbibliotek. Detta beskrivs utförligt i avsnitt 5.4.5 på sidan 195, som du bör läsa ordentligt innan du ger dig på att kompilera program på egen hand.

8.3 GNU Autotools

GNU Autotools är ett antal verktyg som utvecklats inom GNU-projektet med syfte att förenkla och standardisera konfiguration av källkodspaket före kompilering samt själva kompileringen, d.v.s. byggandet av exekverbara program och programbibliotek från källkodsfiler.

Syftet med detta avsnitt är inte att lära ut GNU Autotools för programmerare, utan att förklara dess funktion för den som behöver veta något om dessa verktyg för att kunna kompilera program som anländer i form av källkodspaket.[7]

GNU Autotools består av:

GNU Autoconf — ett konfigureringsverktyg för att generera ett skript med namnet **configure**, som i sin tur kan konfigurera källkodspaket precis innan de kompileras.

GNU Automake — ett verktyg för att automatiskt generera "Makefiler", d.v.s. filer med namnet *Makefile* eller *Makefile.in*.

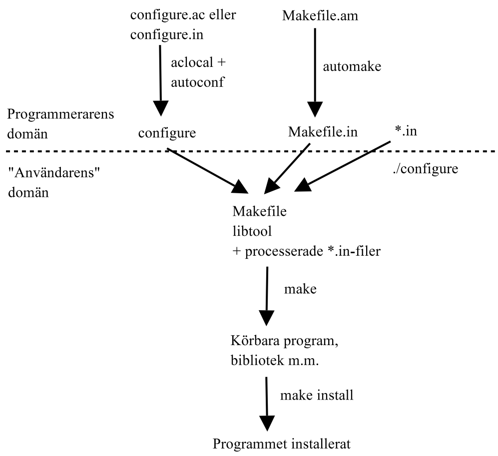
GNU Libtool är ett flertal mindre skript som i sin tur genererar ett skript med namnet **libtool** vid konfigurering av källkodspaket och som i sin tur tar hand om kompilering och installation av dynamiska länkbibliotek på olika plattformar.

Det är möjligt för en programmerare att använda vart och ett av dessa program var för sig, men de flesta använder alla tre samtidigt, och brukar då säga att de kör GNU Autotools som *byggsystem*. Den smidiga möjligheten att konfigurera ett program bara genom att skriva **./configure** kommer från GNU Autoconf, och är egentligen den enda del av Autotools som skall beröra en användare av ett POSIX-system direkt.

8.3.1 Utvecklarens perspektiv

I figur 8.1 återfinns en skiss av flödet mellan de olika auto-verktygen: programmeraren använder tre olika program: **aclocal**, **autoconf** och **automake**.

[7] Den utvecklare som vill lära sig GNU Autotools på djupet rekommenderas att läsa den beryktade "Getaboken", *GNU Autoconf, Automake and Libtool*[36]



Figur 8.1: Arbetsflödet i GNU Autotools. De delar av programmen som du som användare skall använda finns under den streckade linjen.

autoconf används för att kompilera ett program skrivet i skriptspråket GNU M4 till ett stort skript helt och hållet skrivet i Bourne Again Shell (**bash**). M4-programmet lagras i en fil som tidigare hette *configure.in* men som i nyare projekt oftast heter *configure.ac*. Namnbytet skedde för att inte blanda ihop denna *.in*-fil med andra *.in*-filer (som vi skall bekanta oss med senare).

När en programmerare skall kompilera sitt M4-program skriver denne först **aclocal** för att kopiera en del M4-program som ligger lagrade i systemet till en form av programbibliotek. Detta programbibliotek lagras i en fil med det upplysande namnet *aclocal.m4* i programmets byggkatalog.

Själva kommandot **autoconf** kommer sedan att kompilera *configure.in/configure.ac*, bland annat med hjälp av de makron som finns i *aclocal.m4*, till en fil som heter bara **configure**. Detta är den fil som du som användare sedan kör för att konfigurera ett källkodspaket.

En del användare, som gett sig i kast med att titta efter vad skriptfilen **configure** i själva verket *innehåller*, har storögt frågat sig vilken demon som skrivit detta totalt genomröriga och oläsliga skript. Det är alltså inte skrivet av någon människa, utan det är fråga om ett skript som är genererat från ett helt annat program skrivet i språket M4. GNU Autoconf kompilerar alltså M4-program till skript.

Programmeraren kör också **automake** för att generera s.k. makefiler (filer som innehåller kommandosekvenser till kommandot **make**).

Kommandofilerna till **automake** har namnet *Makefile.am*. Det kan finnas flera sådan filer i en källkodskatalog, men den som ligger längst ner i källkodskatalogen pekar i sin tur ut andra *Makefile.am*-filer som skall bearbetas, så kommandot behöver bara köras en gång.

Vanligtvis genererar **automake** inte direkt den make-fil med namnet *Makefile* som används för att slutgiltigt kompilera programmet, utan en "mellanfil" med namnet *Makefile.in*. Denna måste sedan bearbetas av **configure**-skriptet för att till sist producera den fil med namnet *Makefile* som i slutänden faktiskt kompilerar programmet.

Om en utvecklare glömt att köra de här kommandona innan programmet distribuerades ut, eller om du t.ex. hämtar ut källkod från CVS, kan en färdig **configure**-fil saknas. Om det däremot finns en *Makefile.am* och *configure.in* eller *configure.ac* kan du själv skapa **configure**-filen med följande kommandosekvens:

```
# aclocal
# automake
# autoconf
```

Ibland finns ett skript med namnet **autogen.sh** som utför samma sak fast på ett lite mer sofistikerat sätt.

Om nu inte detta fungerar är det inte mycket mening att fortsätta kämpa. Du måste ge upp och antingen hitta en annan lösning eller utbilda dig själv till programmerare.

8.3.2 Användarens perspektiv

Ur användarperspektiv finns det alltid en färdig **configure**-fil att köra. Denna används för att konfigurera källkodspaketet för den dator och det operativsystem där det till sist skall kompileras och användas. Anledningen till att detta behövs är att programmet måste göra anpassningar av en del småsaker som skiljer sig mellan olika operativsystem, samt att beroenden av programbibliotek måste kontrolleras innan kompilering sker.

Du kan också själv behöva göra en del inställningar med **configure**. För att se vilka inställningar som kan göras skriver du **./configure --help**.[8] Detta kommer att lista en rad möjliga val, varav vissa är standardiserade val som alltid går att göra, och vissa (som ofta räknas upp i slutet) är specifika för just detta källkodspaket.

De vanligaste standardinställningarna är:

[8] Du borde vid det här laget veta att **./configure --help | less** ger dig hjälptexten en sida i taget. (Jag bara kollar att du är med.)

./configure --prefix=/foo/bar anger ett *prefix* för vart programfilerna och andra delar av programmet skall lagras. Om du inte anger detta kommer programmet att installeras i */usr/local*-hierarkin, d.v.s. i */usr/local/bin*, */usr/local/lib* och liknande.

En vanlig variant är att du som växel anger miljövariabeln **$HOME** (med **--prefix=$HOME**) för att tala om att programmet skall installeras i din egen hemkatalog. Detta är bra på system som du inte administrerar själv (t.ex. i en skola) men som du ändå vill kunna installera program på. Det skall väl tilläggas att större program sällan är möjliga att installera på detta vis utan en hel del extra slit.

Om du till äventyrs vill kompilera program som skall ligga direkt i */bin*, */lib* eller */sbin* måste du tänka på att ange detta till autoconf med **./configure --prefix=/**. Program som skall paketeras för att vara en del av en distribution skall kompileras med **./configure --prefix=/usr**. (Att installera på dessa platser rekommenderas *inte* om du inte vet exakt vad du håller på med!)

./configure --exec-prefix=/foo/bar talar om att de exekverbara programmen (inklusive programbibliotek) skall installeras på ett speciellt ställe. Detta är ganska obskyrt, men en användare kan t.ex. tänkas använda en och samma hemkatalog för Solaris och GNU/Linux-system, och installerar då kanske körbara Solarisprogram och körbara GNU/Linux-program på två olika platser i sin hemkatalog.

./configure --program-prefix=foo kommer att göra så att alla exekverbara program får ett eget prefix, vilket normalt är meningslöst eller dumt, men om du installerar t.ex. GNU Make på ett system som redan har ett eget **make**-kommando, vill du kanske kalla GNU Make för **gmake** istället, och anger då **--program-prefix=g**.

När du kör **configure** kommer det ibland, när det program du tar dig för att kompilera innehåller programbibliotek, även att generera skriptet **libtool**. Detta skript används för att installera programbibliotek på just ditt operativsystem.

Det sista **configure** gör är att modifiera ett antal angivna filer med namn som slutar på *.in*, så att *Makefile.in* blir *Makefile*, *foo.in* blir *foo* o.s.v. Oftast är det bara ett fåtal filer som påverkas. Den genererade *Makefile*-filen är dock nödvändig för att programmet alls skall gå att kompilera, och måste alltid genereras.

Efter att paketet är konfigurerat skall det kompileras och du skriver **make**. GNU Make använder då *Makefile*-filen (vilket strikt sett är

Miljövariabel	Användning
$CC	C-kompilator. Normalt väljs GCC automatiskt, men har du köpt eller programmerat en ovanligt exklusiv C-kompilator kan du byta den genom att ändra denna miljövariabel. Du kan också ha två eller fler olika versioner av GCC installerade för att kunna byta till en nyare med t.ex. **export CC=gcc4** om det finns en **gcc4**-kompilator installerad vid sidan om din standardkompilator. Denna används normalt även som länkare.
$CFLAGS	Flaggor till C-kompilatorn.
$CPP	Preprocessor för C-kod.
$CPPFLAGS	Flaggor till C-kodspreprocessorn.
$CXX	C++-kompilator. Samma resonemang som för $CC gäller för denna.
$CXXFLAGS	Flaggor till C++-kompilatorn.
$CXXCPP	Preprocessor för C++-kod.
$CXXLINK	Länkare för C++-kod.
$LDFLAGS	Flaggor till länkaren, som bland annat kopplar den kompilerade koden till eventuella programbibliotek. Om du vill kompilera ett program som inte skall vara beroende av några dynamiska länkbibliotek kan du göra detta genom att sätta variabeln till **-all-static**. Detta görs t.ex. för alla program som installeras direkt i */bin*, eftersom de skall kunna fungera självständigt.
$SHELL	Anger standardskalet som skall användas av **make**. Om detta inte anges kommer ditt aktuella skal att användas, vilket inte alltid fungerar som avsett. Prova då att sätta $SHELL till */bin/sh*.

Figur 8.2: Miljövariabler som påverkar beteendet hos kommandot **make**.

ett program skrivet i ett funktionellt språk) för att undersöka vilka filer som skall kompileras, och framför allt i vilken ordning de skall kompileras. Beteendet hos **make** kan modifieras genom att du ändrar någon eller några av miljövariablerna i figur 8.2 innan du skriver **make**.

Efter att **make** har kört färdigt har programmet kompilerats från källkod till körbara program. Då kan det installeras på avsedd plats (*/usr/local* eller den plats som valdes med --prefix=...-flaggan till **configure**) med **make install**. Detta kommer vanligen att anropa ett program som heter **install** och som har till uppgift att kopiera de olika filerna dit de skall hamna, t.ex. i olika underkataloger till prefixet. Detta tar även hand om att sätta normala rättigheter på filerna, så att t.ex. exekverbara filer verkligen är exekverbara.

Även om du kan kompilera ett helt källkodspaket utan att vara root, kan du inte installera ett program annat än i din egen hemkatalog om du inte först växlar till root-användaren. Den plats där programmen

kommer att installeras om du inte angivit något speciellt är */usr/local*, och denna katalog är bara skrivbar för root.

8.4 Att kompilera Linuxkärnan

> **Faust:** Det var således pudelns kärna! En vandrande skolast!
> Jag måste skratta gott.
>
> **Mephistopheles:** Jag hälsar er, bland lärde män en stjärna!
> Av er har jag ett duktigt svettbad fått.
>
> — Johann Wolfgang von Goethe[9]

Att kompilera Linuxkärnan är ett nödvändigt ont för den avancerade användaren. En del lär sig till och med att tycka om detta.

Genom att själv kompilera om kärnan kan du utforma ett system som är specialanpassat till just de komponenter som sitter i just din dator. För ett inbyggt system är denna höga grad av anpassning en självklarhet, men för en vanlig skrivbordsdator brukar detta moment helt utgå, eftersom de flesta distributioner tillhandahåller en färdig kärna, som inkluderar alla tänkbara moduler, som sedan bara laddas in om de behövs. Kärnan tar på det viset inte upp onödigt stort utrymme i datorns minne.

Det finns dock en del situationer när två komponenter i kärnan måste väljas uteslutande av varandra. Ett vanligt sådant dilemma var i 2.4-kärnan valet mellan APM, *Advanced Power Management* och ACPI, *Advanced Configuration and Power Interface*. Dessa två strömbesparningstekniker kunde inte båda finnas i kärnan samtidigt. De kunde inte heller användas i form av moduler. Därför kom de flesta distributioner med det vanligare APM inkompilerat, och ACPI avstängt. I 2.6-versionen av Linuxkärnan avhjälptes problemet genom att göra det möjligt att använda båda funktionerna alternativt med varandra: den modul som laddas in först kommer att användas.

Om du nu bara har en ACPI-dator och en installerad standarddistribution med 2.4-kärnan som inte kan hantera detta, krävs det att du sätter dig ner och kompilerar om kärnan.

Det finns också situationer då någon ny hårdvara precis har dykt upp, och då kan du behöva använda någon experimentell modifika-

[9] Viktor Rydbergs översättning från femte upplagan 1915 av följande originaldialog: **Faust:** Das also war des Pudels Kern! / Ein fahrender Skolast? Der Kasus macht mich lachen. **Mephistopheles:** Ich salutiere den gelehrten Herrn! / Ihr habt mich weidlich schwitzen machen.

tion av Linuxkärnan. Ytterligare en tänkbar situation är att du vill införa kryptering i en äldre version av kärnan (2.4-serien) som inte direkt stödjer detta. Många kompilerar om kärnan för att lägga på extra patchar för realtidsstöd, vilket är bra för t.ex. vissa professionella musiktillämpningar.

Det är dessutom så att den som kompilerat om kärnan några gånger får en god känsla för hur själva kärnan fungerar inuti, även utan att själv vara programmerare.

För att få tag på Linuxkärnan kan du ladda ned den från dess hemsida,[10] men vanligtvis medföljer det ett källkodspaket till din distribution. Om du vill ha din nuvarande konfiguration som utgångspunkt bör du använda distributionens egna källkodspaket, eftersom de inställningar som distributören har gjort oftast finns med som exempel.

Källkoden för kärnan brukar läggas i katalogen */usr/src/linux-x.y.z* där *x*, *y* och *z* är versionen av kärnan. Du kan på det viset ha källkod till flera olika versioner av kärnan liggande. Kompileringen av kärnan sker också i denna katalog.[11]

Förutsatt att du först fått tag i källkoden och ställt dig i den uppackade källkodskatalogen kan du sätta igång med att konfigurera kärnan.

Eftersom en Linuxkärna innehåller så många komponenter och valalternativ går det inte att använda GNU Autotools för att bygga kärnan. Istället används normalt ett menysystem där den som skall kompilera kärnan själv väljer vilka komponenter som skall vara med och vilka som inte skall vara det.

make menuconfig startar ett konfigureringsgränssnitt med menyer som bara använder textläget, och alltså går att köra i nästan vilken terminal som helst. Fordrar att programbiblioteket *curses* (och dess eventuella utvecklingspaket) finns installerat.

make xconfig startar ett fullgrafiskt *X*-baserat konfigurationsgränssnitt. I 2.6.x-kärnan används programbiblioteket Qt i detta alternativ, och detta (samt motsvarande utvecklingspaket) måste finnas installerat.

make gconfig (finns bara fr.o.m. version 2.6.x av kärnan) är ett annat grafiskt konfigurationsgränssnitt, som använder GTK+ för lite snyggare konfigurationsmenyer. Fordrar GTK+ och dess eventuella utvecklingspaket.

---

[10] Se: `http://www.kernel.org/`

[11] Det är möjligt att kompilera objektfiler till en separat katalog om du skriver **make O=/foo/bar**, men notera att detta måste skrivas efter *varje* **make**-kommando.

make config fungerar när inget annat fungerar. Använder bara text, men du måste också svara på hundratals frågor.

När det gäller att välja moduler måste du ha en mental modell av hur Linux fungerar för att kunna välja rätt: vissa delar skall kompileras in i kärnan för att alltid vara tillgängliga och vissa delar skall eventuellt ligga som moduler. Läs noga igenom avsnitt 5.4 på sidan 179 om arkitekturen i GNU/Linux, så att du är medveten om vad du gör.

När du valt dina moduler är det dags att kompilera kärnan. I version 2.4 skriver du då:

```
make dep
make bzImage
make modules
```

Medan du i version 2.6 av kärnan kan nöja dig med:

```
make
make modules
```

Efter detta har kärnan respektive modulerna byggts var för sig.

På vissa distributioner är det möjligt att skriva **make install** för att installera den nya kärnan. I annat fall får du manuellt kopiera den nya kärnan på plats. Själva kärnan ligger på en vanlig PC i *arch/i386/boot/bzImage* i källkodskatalogen och skall kopieras till din boot-partition där den ofta ha namn i stil med */vmlinuz*, */boot/vmlinuz*, */boot/vmlinuz-2.x.x*. Den initiala RAM-disken (om en sådan används) måste, om den används, skapas och installeras separat.[12]

Nu kommer en stor varning.

Kopiera inte bara över din gamla kärna rakt av! Om den nya kärnan inte skulle fungera så sitter du vackert med en dator utan fungerande kärna, och därmed också helt utan fungerande operativsystem. I så fall är enda utvägen att starta systemet från diskett eller CD-skiva, montera hårddiskens filsystem, och återskapa kärnan på något vis. Detta är kanske inte alltid ens möjligt!

Av den anledningen skall du antingen göra en säkerhetskopia av din gamla kärna, eller använda bootstrap-loaderns inbyggda möjlighet att starta flera olika kärnor, så att du lätt kan gå tillbaka till din gamla kärna.

Modulerna behöver sedan installeras separat. Här duger **make modules_install** för det mesta alldeles utmärkt. Detta kopierar filerna på

[12]Red Hat / Fedora Core har ett specialkommando för detta, **mkinitrd**. Om detta inte finns tillgängligt måste denna RAM-disk skapas för hand, genom loopback-montering.

plats i */lib/modules/<kärnversion>*. En ny modulkatalog kommer att skapas för varje unik version av kärnan, med utgångspunkt från namnet varifrån kärnan kompilerades.

Den statiska delen av kärnan kommer sedan att veta om vilken underkatalog den skall läsa sina moduler ifrån. För att göra en experimentell version av den kärna som används i systemet kan du helt enkelt kopiera källkodskatalogen eller byta namn på den, t.ex. **cp -R linux-2.6.4 linux-2.6.4-test**. Om du gör detta kommer en ny modulkatalog med namnet */lib/modules/2.6.4-test* att skapas, och denna gör så att din testkärna inte stör den "riktiga" systemkärnan.

KAPITEL 9

Internet och andra nätverk

När det gäller Internets historia är denna så pass mycket allmänbildning att jag inte kommer att gå in på den här. Den bästa källan till en kort och bra beskrivning av Internets historia är *A Brief History of the Internet*[1] som publicerats av The Internet Society.

Om lokala nätverk i allmänna ordalag kan sägas att de här i landet ursprungligen var den naturliga förlängningen av stordatorsystem enligt klient-server-modellen med speciella seriekablar kopplade från en stordator ut till terminaler.

Under 1980-talet ökade lokala nätverk i popularitet, eftersom de gjorde det möjligt för användare att lagra filer och göra säkerhetskopior centralt. I vissa fall användes inte ens en central filserver: användarna kunde då från en viss dator se innehållet i andra filsystem på andra datore anslutna till nätverket och på så vis utbyta filer.

Sverige började anslutas till olika internationella publika datornätverk kl 14:02 den 7 april 1983,[2] då ENEA Data[3] i Täby anslöts till EUnet av Björn Eriksen. ENEA blev snart navet i ett mindre UNIX-nätverk baserat på protokollet UUCP (UNIX-to-UNIX-copy) omfattande de största svenska universiteten.

År 1986 registrererades toppdomänen *.se* för Sverige, och runt 1987-1988 skapades SUNET (Swedish University Computer Network) och

---

[1] Titeln är en ordlek med Stephen Hawkings boktitel *A Brief History of Time* eller *Tiden, en kort historik* som den hette på svenska. Se [1]

[2] Den första svenska domänen, *enea.se* är också registrerad detta datum.

[3] ENEA var en gång en förkortning av Engmans Elektronik Aktiebolag, men är väl numera antagligen bara varumärket ENEA.

NORDUNET (Nordic University Computer Network). Dessa universitetsnät övergav till sist de gamla nätverkssystemen X.25, DECNet och UUCP för att uteslutande använda internetprotokollet, mest känt som IP. I och med att IP-adresser började användas blev Svenska datornätverk successivt en del av det publika Internet.

9.1 Allmänt om datornät

Om du någon gång har varit i kontakt med ämnet datornät kommer detta avsnitt att tråka ut dig. Hoppa i så fall direkt till nästa avsnitt.

Datornät är nätverk av datorer, d.v.s. datorer anslutna till varandra med en digital informationskanal. De olika anslutningsformerna har gett upphov till följande terminologi:

- Datornät kan klassas efter *storlek*; *lokala nätverk*, *stadsnät* eller "stora nätverk" (Internet).[4]
- De kan klassas relativt en organisation såsom ett företag: ett *intranät* finns internt på företaget, medan ett *extranät* når företagets kunder och leverantörer.
- De kan delas in efter hur kablarna är dragna, d.v.s. enligt topologi: *stjärnnät*, *maskformat nät*, *bussnät*, *ringnät* o.s.v.
- De kan även klassas efter komponenternas roller: klient-server,[5] peer-to-peer (P2P) o.s.v.

Datorer i nätverk använder *protokoll*, vilket är konventioner för hur data skall formateras och kapslas in i *paket*,[6] även kallade *datagram*, bestående av binära talsekvenser, för transport över nätet. Dessa talsekvenser omvandlas och bearbetas sedan som andra (större, komplexare) paket i ett antal nivåer, och till sist förvandlas de till elektriska signaler, ljuspulser, radiovågor eller något annat elektromagnetiskt vågutbredningsfenomen, för att sedan omvandlas tillbaka till paket i andra änden av förbindelsen.

Protokollen innehåller snillrika funktioner för att med hjälp av kontrollsummor säkerställa att paketen kommer fram intakta, och kan ibland

[4] Mer eller mindre godtagna översättningar av engelskans LAN (Local Area Network), MAN (Metropolitan Area Network) och WAN (Wide Area Network).

[5] Se avsnitt 2.3.10 på sidan 104.

[6] Paketförmedlade datornät är den idag dominerande formen av datornätverk. Principen är inte annorlunda från att skicka paket med posten eller någon budfirma, med skillnaden att det rör sig om abstrakta enheter, som kan betraktas som komponenter av ett större meddelande.

Tillämpning: FTP, HTTP, SSH o.s.v.
⇕
Transport: TCP, UDP, RTP
⇕
Nätverk: IP, ARP, RARP, ICMP
⇕
Länk: PPP, SLIP, Ethernet, Token ring o.s.v.

Figur 9.1: TCP/IP-stacken har fyra tydliga lager: tillämpning, transport nätverk och länk. Under länken finns sedan de fysiska elektromagnetiska signalerna.

även se till att de kommer fram i en viss ordning, eller åtminstone att de sorteras när de kommer fram.

De protokoll som med förkrossande majoritet används av dagens nätverk brukar kallas *internetprotokoll*. De har tack vare Internets framväxt fått en sådan total dominans att de i princip helt ersatt gamla protokoll som IPX/SPX[7] från Novell, X.25 från ITU-T, AppleTalk från Apple och NetBEUI från Microsoft. Internetprotokollen, som vanligen bara kallas "TCP/IP-stacken",[8] har det hierarkiska utseendet i figur 9.1.

De olika hierarkiska stegen brukar i datorkommunikationssammanhang kallas för *lager*. Lagren i TCP/IP-stacken heter således tillämpningslager, transportlager, nätverkslager och länklager.[9]

Nuvarande standardiseringsorganisationen ITU-T[10], före detta standardkommittén CCITT,[11] har uppfunnit en modell som kallas OSI (Open Systems Interconnect) och har vissa vaga likheter med realiteterna i TCP/IP-stacken. Här har samtliga inblandade protokoll namn som börjar med **X** (X.25, X.500 o.s.v.) och systemet avsåg att ersätta alla befintliga protokoll, inklusive det beprövade och tålmodigt utarbetade TCP/IP. Detta projekt har numera vederbörligen dött sotdöden, men används ändå ibland för att t.ex. tortera studenter som läser datorkommunikationskurser på universitet.

Linuxkärnan är, liksom de flesta POSIX-system, byggd för att användas tillsammans med TCP/IP. Kärnan *kan* naturligtvis hantera andra protokoll som IPX eller AppleTalk, men nyttan med detta är nog

[7] IPX/SPX utläses *Internetwork Packet eXchange/Sequenced Packet eXchange.*

[8] Transmission Control Protocol/Internet Protocol är ett samlingsnamn på protokollstacken.

[9] Lagerterminologin ger upphov till en del facktermer av typen "lager-3-router". Undvik helst sådan jargong, eftersom det ofta är oklart vilket lager som är det tredje.

[10] ITU-T utläses Internationella teleunionen, Telekom-avdelningen.

[11] CCITT utläses Comité Consultatif International Téléphonique et Télégraphique.

ganska begränsad.

Från länklagret och nedåt kan kärnan hantera alla möjliga protokoll. Ethernet och PPP (point-to-point protocol) är kanske de vanligaste, men FDDI,[12] SLIP,[13] Token Ring, och ATM[14] kan lika gärna användas. På senare tid har även lokala radionätverk (engelska: *wireless local area network*, WLAN) blivit populära och kärnan stöjer även dessa protokoll.

9.2 TCP/IP-stacken: internetprotokollen

Detta avsnitt avhandlar internetprotokollstacken, som används av de flesta moderna datorsystem, och av alla system som är anslutna till Internet. Om du tidigare använt TCP/IP i ett annat operativsystem är detta avsnitt inga nyheter eftersom protokollen inte skiljer sig mellan operativsystemen, och du kan med gott samvete hoppa direkt till konfigureringsavsnittet, som går in på hur just GNU/Linux använder TCP/IP.

De protokoll som ingår i TCP/IP-stacken definieras av Internet Engineering Task Force, IETF. Detta arbete sker genom arbetsgrupper där vem som helst kan bli medlem genom att prenumerera på arbetsgruppens epostlista. Arbetet dokumenteras i standarddokument med namngivning av typen *RFC NNNN*, av engelskans *Request For Comments* och där *NNNN* är en siffra.

Namnet *Request For Comments* kan låta bakvänt och instabilt, men har historiska skäl och dokumenten har hetat så sedan 1969. RFC-dokumenten är förvånansvärt lättlästa jämfört med många andra tekniska standarddokument.[15]

Med figur 9.1 i gott minne skall vi som hastigast bekanta oss med de viktigaste protokollen i TCP/IP-stacken. För att förstå dessa protokoll fordras kunskaper i binär aritmetik (i synnerhet att kunna översätta mellan binära och decimala tal). Om du inte har sådana kunskaper kommer en del av det följande att framstå som mystiskt och obegripligt, och du kan då nöja dig med att tillsvidare betrakta IP-nummer, nätmask, standardgateway och DNS som något magiskt som du får från din nätverksleverantör. Många får alla sina nätverksinställningar via DHCP (se nedan) och behöver därför inte bekymra sig om saken.

[12]FDDI, vilket står för *Fibre Distributed Data Interface* är ett lågnivåprotokoll avsett för lokala nätverk byggda på optisk fiber istället för metallkablar. Det kan dock lika gärna användas på metallkabel och kallas då CDDI, *Cable Distributed Data Interface*.

[13]SLIP utläses *Serial Line Interface Protocol* och är en föregångare till PPP.

[14]ATM står för *Asynchronous Transfer Mode* och används mycket i högpresterande förbindelser.

[15]Se: `http://www.ietf.org/rfc.html`

Det som följer är på inget vis någon uttömmande redogörelse för internetnätverk: rekommenderad läsning för fördjupning är motsvarande RFC-dokument, som i vart fall är angivet.

9.2.1 Tillämpningsprotokoll

Tillämpningsprotokollen, varav det finns ganska många, skall här enbart avhandlas sporadiskt: de används på hög nivå direkt av olika tillämpningsprogram för att kommunicera med andra program. De allra flesta av dessa protokoll består i praktiken av några rader med ren text, följd av en dataström. För att sedan sköta själva transporten av dataströmmen använder de vanligtvis TCP, UDP eller RTP, som alla beskrivs nedan.

Samtliga tillämpningsprotokoll bygger på klient-server-modellen, se avsnitt 2.3.10 på sidan 104.

HTTP *Hyptertext Transfer Protocol* (RFC 2068) skapades ursprungligen av Tim Berners-Lee vid CERN för transport av HTML-dokument över Internet. HTTP öppnar en TCP-förbindelse, hämtar eller skickar data, och stänger sedan förbindelsen igen. HTTP kan också återuppta en påbörjad överföring som av någon anledning avstannat, och används därför ofta även till filöverföring. Se vidare avsnitt 11.2 på sidan 379 om webbläsare.

FTP *File Transfer Protocol* (RFC 959) är ett protokoll som överför hela filer över Internet. Ett FTP-program öppnar en TCP-förbindelse till en annan dator, och håller den sedan öppen tills användaren har hämtat alla filer som denne behöver.

SMTP *Simple Mail Transfer Protocol*[16] (RFC 2821) används för att skicka elektronisk post över Internet. Det motsvarar postverkets gula brevlådor, där posten stoppas i för att skickas vidare till mottagaren. SMTP använder sig i sin tur av TCP.

IMAP *Internet Message Access Protocol* (RFC 3501) används för att ta emot elektronisk post från en brevlåda dit det levererats med protokollet SMTP. Det motsvarar den vanliga brevlådan vid grinden till ett hus eller i en lägenhetsdörr. Ett äldre protokoll för epost kallas **POP** *Post Office Protocol* (RFC 1939 m.fl., även kallat POP3) och används allt mindre. IMAP kan hantera mycket som POP inte kan, t.ex. lagring av brev på en server, till skillnad från POP som bara kan hämta hem dem till din egen dator. Både IMAP och POP använder TCP.

[16] Ibland skämtsamt kallat ”Send Mail To People”.

NTP *Network Time Protocol* (RFC 1305 och RFC 2030) används för att synkronisera klockan på en dator mot klockan på en annan dator som erbjuder en synkroniseringsserver. NTP använder i sin tur vanligtvis UDP.

SSH *Secure Shell* är ännu ingen standard, eller ens föreslagen standard. Endast ett utkast till standard föreligger. Emellertid används SSH mycket för att via TCP öppna en krypterad terminalförbindelse mellan två datorer, och är därför *de facto*-standard. SSH ersätter tidigare protokoll som **Telnet** (RFC 854 och RFC 855) som inte erbjöd någon krypering alls, utan bara ett ganska simpelt terminalfönster. Terminalinloggning är för övrigt Internets äldsta praktiska användingsområde och den direkta anledningen till att föregångaren ARPAnet en gång skapades.

Att starta Internettillämpningar (även kallat tjänster) på en dator är att göra datorn till en *server* för en viss tjänst. T.ex. är en dator med en demon för protokollet HTTP en dator som kan tillhandahålla tjänsten "hypertext" och kallas således för en "webbserver".

Att starta servertjänster utöver de mest grundläggande betraktas i denna bok som stordrift (ej husbehov), och vilka GNU/Linux-program som används för dessa tjänster avhandlas endast översiktligt i appendix B.4 på sidan 423.[17]

Tillämpningsprotokoll: DNS

Domain Name System (RFC 1034 och RFC 1035)[18] är ett viktigt protokoll: detta används för att koppla samman ett symboliskt namn på en dator med ett IP-nummer (vilket kommer att förklaras senare, men som i princip är ett slags "telefonnummer" till en annan dator), så att t.ex. datornamnet *www.foo.org* automatiskt översätts till IP-numret 123.124.125.126.

DNS är ett tillämpningsprotokoll, men upplevs lätt som om det låg "längre ner" i nätverksstacken, eftersom det används av i princip alla tillämpningar. T.ex. använder varje program som kan utnyttja FTP också DNS, varje webbläsare som använder HTTP använder också DNS o.s.v.

[17] Annan litteratur på området tvekar inte att introducera både FTP- och HTTP-servrar för nybörjare, möjligen i tron att många som använder GNU/Linux vill använda systemet till just sådant. Skulle sådana program och konfigurationer behandlas på ett rättvist sätt skulle den här boken vara minst dubbelt så tjock.

[18] RFC 1034 och 1035 utgör en uppdatering av den ursprungliga standarden som publicerades i RFC 882 och RFC 883.

I begynnelsen fanns endast IP-nummer. Numren var svåra för användarna att komma ihåg, och därför började symboliska namn att användas. Bruket av symboliska adresser tog sin början i att datorerna anslutna till Internet använde en fil kallad *hosts* som fanns på varje dator, och som innehöll listor över vilka datornamn som svarade mot vilka IP-nummer. Alla datorer på hela Internet fanns uppräknade i denna fil. Filen finns fortfarande i de flesta POSIX-system, och i GNU/Linux hittar du den i */etc/hosts*. (Vi återkommer till *hosts*-filen längre fram.)

Efter ett tag blev det omöjligt att underhålla den här filen. Folk hade inaktuella och felaktiga *hosts*-filer, och ibland manipulerades filerna godtyckligt av administratörer. För att lösa detta skapades det hierarkiska, distribuerade, självreplikerande och eleganta DNS-systemet, och med det alla de kryptiska Internet-adresser vi använder idag.

Datornamnen som anges i DNS är delade i två delar: *datornamn* (engelska: *hostname*) och *domännamn* (engelska: *domain name*). Om de två delarna slås samman med en punkt kallas resultatet för det *fullständigt kvalificerade domännamnet* (engelska: *fully qualified domain name*, FQDN).

Datorn med det fullständigt kvalificerade domännamnet *www.foo.org* har exempelvis datornamnet *www* och domännamnet *foo.org*.

9.2.2 Transportprotokoll: TCP

Transmission Control Protocol (RFC 793) fastslogs redan år 1981. TCP packar in data i paket med en storlek mellan 25 och 65536 byte. Cirka 90% av all trafik på Internet består av TCP-paket.

Varje paket har ett sekvensnummer: detta används för att på mottagarsidan se till att paketen kommer fram i rätt ordning och för att detektera om något paket skulle saknas och i så fall begära omsändning av paketet.

Med hjälp av sekvensnumren och omsändningsmekanismen kan TCP-paketen passas samman till en förlustfri dataström av samma slag som en fil. Det finns även en kontrollsumma i paketet som kan användas för att verifiera att paketet överförts korrekt.

Förutom ren data innehåller ett TCP-paket även två *portnummer*: porten paketet skickades från och porten det är avsett att gå till. Dessa 16-bitarssiffror är rena konventioner och har inget att göra med portar på baksidan av datorn eller i datorns minne. De används för att flera tillämpningsprogram skall kunna använda TCP samtidigt: om detta inte görs kan mottagare och sändare inte avgöra vilken process i operativsystemet[19] som skall hantera ett inkommet paket.

[19]Den mottagande processen (programmet) är oftast en demon (t.ex. en server, se si-

Port	**Protokoll**	**Port**	**Protokoll**
20, 21	FTP	22	SSH
23	Telnet	25	SMTP
53	DNS	69	TFTP
70	Gopher	79	Finger
80	HTTP	88	Kerberos
110	POP3	119	NNTP
123	NTP	143	IMAP
177	XDMCP	540	UUCP

Figur 9.2: Ett antal standardiserade portar och deras protokoll. Portnumren tilldelas av IANA (Internet Assigned Numbers Authority) och omfattar oftast transport för det avsedda protokollet över både TCP och UDP, även om TCP i de flesta fall är det enda som aktivt utnyttjas.

Genom att kombinera kunskap om portnummer med ett IP-nummer (se nedan) kan två processer på två olika datorer utbyta information över ett TCP/IP-nätverk, t.ex. över Internet. Denna kombination av nätverksadress och portnummer kallas för en *sockel* (engelska: *socket*).

Vilket portnummer som skall användas till vad är dels stanardiserat av IANA *Internet Assigned Number Authority*, dels baserat på ren konvention. De standardiserade portnumren svarar vanligen mot ett visst tillämpningsprotokoll och därmed mot ett specifikt användartillämpningsprogram. Några vanliga exempel återfinns i figur 9.2. I filen */etc/services* som finns i de flesta distributioner finns en längre och mer utförlig lista över vilka portar som är till för vad.

Notera att dessa portnummer är de som används för *inkommande* trafik. Utgående trafik använder en slumpmässigt vald port i ett högt sifferområde[20] så även om du ansluter till en annan dators port 80 för att hämta en webbsida, kan ditt program mycket väl ha angett att svaret skall skickas till port 49160.

9.2.3 Transportprotokoll: UDP

Uniform Datagram Protocol (RFC 768) är en ”förlustvariant” av TCP. Liksom TCP använder UDP checksumma och portnummer och kan således skapa socklar, men några sekvensnummer finns inte.

---

dan 42) men kan också vara ett interaktivt program som t.ex. en webbläsare eller ett epostprogram.

[20]Olika protokoll har definierat olika områden bland de ”höga portnumren” som de använder för klientanslutningar, medan numren mellan 49192 och 65535 (hexadecimalt $C000-$FFFF) är helt fria att använda hur som helst.

UDP används för att s.a.s. ”kasta data” på en mottagare. Om paketet inte kommer fram eller är trasigt, anses det ha gått förlorat. UDP används ofta för att skicka information som är ”bråttom” och som ändå skulle vara värdelös om den skickades om senare.

Eftersom UDP ofta ges högre prioritet än TCP i nätverken händer det att tillämpningsprogram använder UDP på ett okynnesvis för att ge sig själva högre prioritet. Det går t.ex. att emulera omsändning och resekvensiering av UDP-paket högre upp i lagerhierarkin.

9.2.4 Transportprotokoll: RTP

Real Time Protocol (RFC 1889) är ett protokoll som började utarbetas redan i början av 1980-talet och som omnämns i specifikationen för TCP, men som inte publicerades förrän 1996 och inte förrän nu börjat bli användbart. Det används för strömmande video och ljud, IP-telefoni (även kallat VoIP) och liknande realtidstillämpningar.

9.2.5 Nätverksprotokoll: IP

Internet Protocol (RFC 791) är det viktigaste av internetprotokollen. Såväl TCP, UDP och RTP-paket kapslas in i IP-paket för att kunna skickas till den mottagande datorn. Den viktigaste tilläggsinformationen i ett IP-paket är mottagarens och avsändarens *IP-nummer*, ett slags telefonnummer på Internet som leder fram till den mottagande datorn.

För att IP skall fungera på din dator krävs att din dator känner till följande saker, som kommer att förklaras närmare längre ned i detta avsnitt:

- Sitt eget, unika IP-nummer, t.ex. 192.168.1.64
- Sin subnätmask, t.ex. 255.255.255.0.
- Sin standardgateway, t.ex. 192.168.1.1
- Ofta krävs också adressen till en DNS-server för att din nätverksanslutning skall upplevas som fullt fungerande, men strikt sett krävs inte detta för att själva IP-nätverket skall fungera.

Alla dessa siffror kan tilldelas din dator automatiskt genom något av protokollen PPP eller DHCP. Dessa kommer att förklaras längre fram. Om du inte använder PPP eller DHCP måste du ange dem själv.

IP-nummer används för två saker: att numrera nätverk och att numrera datorer. Det finns naturligtvis fler datorer än nätverk.

IP-numren är 32 bitar (fyra byte) långa, och kan således användas för att numrera hela 2^{32} (drygt fyra miljarder) datorer och nätverk i teorin, men många nummer är reserverade och många nummerserier är utdelade i stora sjok som inte alltid används. De 32 bitarna är delade i tre grupper: bitar i början som anger nätverksklass (se nedan), bitar som anger nätverksnummer och bitar som anger datornummer.

För att adressera själva nätverket används en IP-adress med alla datornummerbitarna satta till 0. För att adressera *alla* datorer på nätverket (en s.k. *broadcast*, betyder ungefär ”rundradio”), används en IP-adress med alla datornummerbitar satta till 1. Nätverket med nummer 192.168.1.0 innehåller alltså datoradresserna 192.168.1.1 till och med 192.168.1.254 och broadcast-adressen 192.168.1.255. Om ett paket av något slag skickas till broadcast-adressen kommer alla datorer i nätverket att lyssna.

Om ditt nätverk har publika IP-nummer och är en del av Internet, är en av datorerna, eller rättare sagt: en av de enheter som har ett IP-nummer, en *gateway*. Denna dator har till uppgift att förmedla paket vidare till Internet. Om ett pakets IP-adress inte kan hittas på ditt lokala nätverk måste det ”skickas vidare uppåt” tills det kommer till rätt nätverk. Detta är anledningen till att datorer som skall ingå i ett IP-nätverk måste förses med en *standardgateway* — det kan nämligen finnas flera gateway:ar[21] och systemet behöver då en partner att vända sig till om ingen annan gateway frivilligt tar på sig att förmedla ett paket till Internet.

IP-adresserna brukar skrivas ut med varje byte av 32-bitarsadressen angiven separat, med byte:arna avskilda med en punkt, t.ex. 192.168.1.64. Eftersom varje siffra representerar en byte kan den bara anta värden mellan 0 och 255.

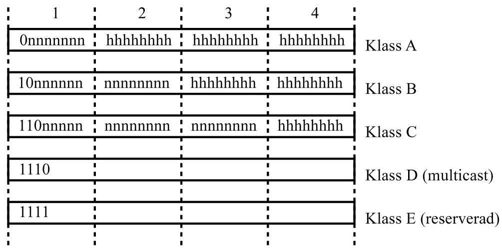
Näten är delade i fem olika klasser: A, B, C, D och E. Klass A-nät är enorma och kan innehålla upp till $2^{24} - 2$ (drygt 16 miljoner) datorer.[22] Klass B-nät omfattar $2^{16} - 2$ (65534 st) datorer och klass C-nät, som är vanligast, kan innehålla 254 olika datorer. Indelningen illustreras i figur 9.3.

Av praktiska skäl används också en s.k. *nätmask* eller *subnätmask*. Denna består av fyra byte (precis som en IP-adress) med alla bitar som adresserar nätverket (de inledande bitarna plus *n*-bitarna i figur 9.3) satta till 1, och de övriga bitarna (som indikerar datorn, *h*-bitarna i figuren) satta till 0. För ett klass C-nät blir subnätmasken t.ex. 255.255.255.0.

---

[21]Svensk terminologi för ordet ”gateway” saknas tyvärr. ”Inkörsport” är den ordagranna översättningen av ordet. ”Förmedlare” är ett tänkbart alternativ.

[22]Notera att två adresser faller ifrån: en för att adressera själva nätverket och en för att adressera alla datorer (broadcast). I praktiken går ofta minst en adress till åt för gateway.



Figur 9.3: Illustration av adresseringsklasserna för IP-nummer. Figuren visar adressernas fyra byte: de bitar som givits värdet 1 eller 0 är fixerade, och variabeln *n* betyder att bitarna används för att numrera nätet, medan *h* numrerar datorerna i nätet (*h* kommer av engelskans *host*, värddator.) T.ex ser vi att det finns $2^7 = 127$ möjliga klass A-nät med 2^{24} datorer i varje, ett av dem används för 127.0.0.1-adressen.

Det främsta användningsområdet för nätmasken är då operativsystemet eller nätverkselektroniken skall avgöra ifall en dator ligger på det lokala nätverket eller ej. En logisk AND mellan datornumret och (sub)nätmasken ger i så fall själva nätverkets adress.

Verkliga *subnät* blir det fråga om ifall bitmasken delar av nätverkets adressrymd i någon jämn tvåpotens. Bitarna längst till vänster i masken måste alltid vara satta till konsekutiva 1:or och bitarna längst till höger måste vara satta till konsekutiva 0:or, men genom att skjuta till en etta till en klass C-nätmask, så att den blir 255.255.255.128, kommer två subnät med 128 datorer i varje att skapas. Om ditt klass C-nät tidigare hade adressen 192.168.1.0 finns nu subnäten 192.168.1.0 med datorer med IP-nummer från 192.168.1.1 till 192.168.1.126 och broadcast-adressen 192.168.1.127 och sedan ännu ett nätverk med adressen 192.168.1.128 och datorer med nummer mellan 192.168.1.129 och 192.168.1.254, och broadcast-adressen 192.168.1.255.

Det vanligaste användningsområdet för subnät är att dela klass B- och klass A-nätverk i mindre bitar. På så vis kan delar av de stora klass A-näten portioneras ut till behövande.

Den typ av nätverksnumrering som använder en subnätmask är inte lämplig för all nätverksutrustning. Vissa routrar[23] kunde t.ex. till en början bara hantera subnätklasserna A, B och C, och därmed bara nätmaskerna 255.0.0.0, 255.255.0.0 och 255.255.255.0. För att lösa detta

[23]*Routrar* är det svengelska namnet på en nätverkskomponent som kopplar samman två eller flera olika nät med varandra. De skiljer sig från en s.k. *hubb*, som inte är stort mer än en grenkontakt för nätverkskablar, genom att routern kan bearbeta inkommande och utgående paket på ett intelligent vis, som datornätets motsvarighet till en telefonväxel.

problem introducerades en notation med namnet *klasslös interdomänroutning* (engelska: *classless inter-domain routing*, CIDR) som anger nätmasken som *antalet ettställda bitar från vänster*, så att t.ex. ett klass C-nät kan ha adressen 192.168.1.0/24. Denna notation är bra att känna till eftersom den ger möjlighet att dela ner nätverk i storlekar andra än A, B och C-klass, så länge nätets storlek kan uttryckas som en jämn tvåpotens.

Vissa *speciella* IP-nummerserier är dokumenterade i RFC 3330. Här listas t.ex. 127.x.x.x som är ett helt klass A-nät reserverat bara för den lokala datorn (som alltid har IP-nummer 127.0.0.1). Det finns också 273 hela nätverk reserverade för "privat användning": 10.x.x.x (ett klass A), 172.16.x.x–172.31.x.x (sexton stycken klass B) och 192.168.x.x (256 stycken klass C). Detta betyder att om du skapar ett lokalt nätverk som inte skall vara en del av Internet, så skall du använda nummer ur t.ex. 192.168.x.x-serien om du maximalt behöver kunna adressera 256 datorer. Detta används mycket.

En enskild dator kan ha mer än ett IP-nummer, men är den ansluten till Internet så har den alltid minst två: 127.0.0.1 och ett unikt, publikt IP-nummer.

Den version av IP-protokollet som dokumenteras i RFC 791 kallas ibland IPv4, eftersom det var den 4:e revisionen av IP, och den som blev utbredd. På grund av att den mängd av IP-nummer som kan åstadkommas med det nuvarande systemet inte räcker till har IETF även specificerat ett nytt protokoll: *IPv6* (RFC 2373 och RFC 2374). Detta används parallellt med IPv4 under en övergångsperiod. IPv6 innehåller flera finnesser, men den viktigaste är att protokollet har stöd för 2^{128} datoradresser, vilket motsvarar cirka 667 kvadriljoner datorer per kvadratmillimeter på jordens yta.[24] Detta borde räcka även för de läskiga maskinerna i filmen The Matrix.

IPv6-adresserna anges hexadecimalt med kolon mellan varje fyrställig grupp, t.ex: 416c:6c20:796f:7572:2062:6173:6520:6172.[25] De första 64 bitarna anger nätverkadressen och de sista 64 bitarna anger datoradressen.

9.2.6 Nätverksprotokoll: ICMP

Internet Control Message Protocol (RFC 792) är egentligen en del av IP-protokollet, och alla implementationer av IP måste även implementera ICMP. Protokollet används t.ex. för att ta reda på vilken väg ett IP-paket

[24] Jordens yta är c:a $5,1 \cdot 10^8$ km^2.

[25] Om flera fyrställiga grupper innehåller värdet 0000 kan de slås samman och uteslutas med en notation som innebär att *en* räcka 0000-grupper kan ersättas med två kolon "::".

skall ta; exempelvis vilken dator som närmast skall skicka paketet vidare. Det används även för program som **ping** och **traceroute** för att testa tillgängligheten till olika datorer på Internet.

9.2.7 Nätverksprotokoll: ARP

Address Resolution Protocol, adressupplösningsprotokollet (RFC 826) är ett protokoll som används på Ethernet-nätverk för att ta reda på närmast mottagande dators MAC-adress. MAC-adressen (från engelskans *Media Access Control address*) är 48 bitar lång och skall vara unik för varje nätverkskort. På vissa kort kan den dock ändras. MAC-adressen hör hemma i länklagret.

För att skicka ett IP-paket över ett lokalt nätverk av typen Ethernet duger det inte med ett IP-nummer: istället fordras en unik adress kopplad till det mottagande nätverkskortet. ARP hjälper till att vaska fram denna adress.[26]

ARP:s tvillingprotokoll RARP *Reverse Address Resolution Protocol* det omvända adressupplösningsprotokollet (RFC 903) används i omvända fallet: du har en MAC-adress och vill veta vilket IP-nummer som tillhör datorn som har denna MAC-adress.

9.2.8 Länkprotokoll: PPP

Point-to-Point Protocol (RFC 1661 och RFC 1662) används för att etablera en nätverksförbindelse mellan två ”punkter”, t.ex. mellan två datorer.[27] Ett underprotokoll till PPP kallas LCP, *Link Control Protocol*, och används för att överföra olika meddelanden om länken mellan de två punkterna. LCP används för att upprätta länken, t.ex. för att överföra ett användarnamn och ett lösenord som släpper in den nya ”punkten” i nätverket.

LCP kan utökas med olika moduler beroende på vad som behöver göras mellan de två punkterna. Den första, och mest självklara modulen är PAP, *PPP Authentication Protocols* (RFC 1334), som gör det möjligt att ange användare och lösenord.

PPP måste inte användas till att ansluta till ett IP-nätverk: vilken nätverkstyp det är fråga om bestäms av ett annat protokoll som är spe-

[26] I själva verket är det så, att när ett IP-paket skickas över Ethernet, packas det i sin tur in i ett Ethernet-paket, förses med den mottagande datorns MAC-adress och skickas ut i ”etern”, d.v.s. nätverkskabeln. Om krockar eller fel uppstår i paketet får TCP eller motsvarande protokoll på högre nivå ta hand om att skicka om paketet tills dess att det kommer fram.

[27] Historiskt sett grundar sig PPP på det äldre protokollet HDLC, *High Level Data Link Control*, men för en nutida användare är väl denna relik i sammanhanget en pipa snus.

cifikt för IP, IPX, DECNET eller vad som nu används i nätverket i fråga. Det som med förkrossande majoritet används är naturligtvis IP, och då är det modulen IPCP — *PPP Internet Protocol Control Protocol*[28] — (RFC 1332) som används. IPCP kan via utökningar tilldela såväl standardgateway, subnätmask och DNS-server till den som ansluter sig.

PPP är ett klient-server-protokoll:[29] en dator agerar *server* som erbjuder *klienter* att ansluta sig till den och erhålla IP-adresser och annan konfiguration via IPCP.

Detta sätt att tilldela information är mycket populärt bland internetleverantörer, eftersom det gör det möjligt att hushålla med IP-adresser. En modempool med 500 modem behöver t.ex. bara ha tillgång till 500 IP-adresser, även om 60000 personer använder modempoolen — bara maximalt 500 personer kan ju använda den samtidigt!

Det vanligaste användningsområdet för PPP är för att ansluta en dator till ett nätverk via modem. Så gott som alltid är det en anslutning till Internet (eller ett internt IP-baserat nätverk) som görs med PPP. Modemet som sitter i din dator ringer upp ett modem som sitter i en annan dator, och via modemlänken förhandlar din dator och den andra datorn med hjälp av PPP och dess underprotokoll om hur kommunikationen skall fungera. Med PAP ger din dator sig till känna, och med IPCP ber den om olika IP-inställningar.

Andra möjliga användningsområden för PPP är att uprätta en förbindelse mellan två datorer över ISDN (Integrated Services Digital Network), över en simpel seriekabel, eller mellan två hela lokala nätverk över en seriekabel eller telefonförbindelse.

Användandet av PPP i Sverige minskar i takt med att ”bredbandsnäten” (vad nu det betyder) byggs ut. Eftersom ordet ”bredband” inte betyder någonting[30] myntar vi begreppet *höghastighetsanslutning* istället, för att referera till abonnemangstjänster med en fast Internetanslutning som inte sker via modem utan via Ethernet. Höghastighetsnät använder i allmänhet DHCP (se nästa avsnitt) för att konfigurera förbindelsen.

Före PPP-protokollets födelse år 1994, och speciellt i Internetboomens ungdom på det tidiga 1990-talet, användes ofta SLIP (*Serial Line Interface Protocol*, RFC 1055) istället för PPP. Detta protokoll kunde t.ex. inte tilldela den uppringande datorn något IP-nummer och är idag inte särskilt vanligt.

---

[28] Det är inte jag som väljer dessa idiotiska namn. Jag ursäktar å IETF:s vägnar.

[29] Se avsnitt 2.3.10 på sidan 104.

[30] Detta modeord har ingen teknisk betydelse. *Bredbandig radiokommunikation* existerar och betyder att radiokanalen tar upp ett stort frekvensområde och således kan överföra stora mängder information. Student-TV i Lund lanserade i ett program begreppet *fetkabel*, vilket säger ungefär lika mycket.

9.2.9 Svårdefinierat: DHCP

Dynamic Host Configuration Protocol (RFC 2131) kan användas för att tilldela IP-adress, standardgateway, nätmask och DNS-server till en nystartad klientdator helt automatiskt. Det fungerar så att en dator som startas i nätverket använder broadcast för att fråga själva nätverket om vilken IP-adress o.s.v. som den skall ha. En speciellt konfigurerad dator — DHCP-servern — kommer då att svara med den begärda konfigureringsinformationen.

DHCP är svårt att placera in i bilden av en nätverksstack: det är i någon mening ett applikationsprotokoll, men det är samtidigt komplicerat att tänka sig en applikation som kan fungera utan att använda de andra lagren i stacken. Funktionsmässigt är det inte svårt att förstå DHCP, och därför rekommenderas en syn på DHCP som något "eget" helt fristående från nätverksstacken.[31]

I ett nätverk där DHCP används utgör det ibland ett säkerhetsproblem: vem som helst som ansluter en dator till nätverket får ju tilldelat sig en IP-adress, och kan då börja använda nätverket! Det är dessutom fritt fram att ansluta hur många datorer som helst till nätet, bara du har tillgång till det fysiska nätverket. Av dessa anledningar villkoras ibland DHCP-tilldelningen så att bara datorer vars MAC-adress registrerats hos DHCP-servern kan tilldelas ett IP-nummer.

I vissa fall uppstår önskan om att en viss dator i ett nätverk som administreras med DHCP skall ha ett fast IP-nummer. Detta fungerar utmärkt — numret kan antingen anges lokalt på datorn, och sedan plockas bort ur den nummerserie som DHCP-servern har tillgänglig att dela ut, eller så kan DHCP-servern konfigureras att tilldela datorn med en viss MAC-adress samma IP-nummer varje gång den begär ett.

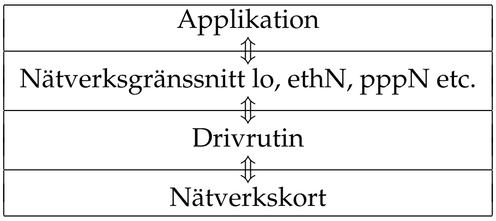
Som redan tidigare nämnts är DHCP populärt på dagens höghastighetsanslutningar.

Tidigare användes protokollet BOOTP (RFC 951) istället för DHCP. Detta har dock gått ur tiden på samma vis som PPP har ersatt SLIP.

9.3 Att ställa in nätverket

Det absolut trevligaste alternativet när det gäller att konfigurera nätverket i GNU/Linux är om din distribution kan gör det åt dig. Detta gäller i synnerhet sådana saker som modemanslutningar och ADSL-linjer.

[31] Jag har märkt att en del människor har stora problem med saker som "inte passar in i modellen". Tänk på att internetprotokollen är utvecklade baserat på praktisk erfarenhet av vad som behövs i ett datornätverk och inte i något platonskt parallelluniversum.



Figur 9.4: Nätverket så som det uppfattas av programmen i GNU/ Linux-sviten. Bland de tillämpningar på översta nivån som talar med nätverksgränssnitten finns *Net-tools*, som sköter alla grundinställningar av nätverket då datorn startas. *Inetutils*-verktygen, SSH, Mozilla samt alla andra nätverksklienter och servrar är andra tänkbara tillämpningar.

Har du ett väl fungerande lokalt nätverk eller t.ex. ADSL-anslutning av något slag, räcker det för det mesta med att någon gång under installationen ange att du vill använda DHCP för nätverkskonfiguration, och sedan behöver du inte bekymra dig mer om saken.

Har du en modemuppkoppling kan det vara knepigare: då krävs ofta en del extra inställningar, t.ex. telefonnummret till din modempool, användarnamnet och lösenordet. Även dessa inställningar klarar de flesta distributioner galant.

Nu går vi in i närbild på hur GNU/Linux använder nätverket. Linuxkärnan och de program som använder sig av den uppfattar nätverket som i figur 9.4.

Här är *ethN*, *pppN* o.s.v. ett numrerat *nätverksgränssnitt*.[32] Din dator kan vara ansluten till flera nätverk med flera olika nätverkskort eller modem. Det är emellertid vanligast att du bara har *en* nätanslutning och således bara *ett* nätverkskort eller ett modem, och då finns det normalt bara endera *eth0* eller *ppp0*. Har du en ISDN-anslutning kallas dess gränssnitt *ippp0*.

Den egna datorn har även ett s.k. loopback-gränssnitt med namnet *lo*, som svarar mot IP-adressen 127.0.0.1 och som inte använder någon hårdvara alls, utan bara pekar anslutningar du gör till den egna datorn tillbaka till dig själv.

Andra typer av nätverksgränssnitt existerar, men de nämnda typerna står för en förkrossande majoritet av alla i naturen förekommande nätverksanslutningar.

Bakom det skikt i modellen som bara kallas *nätverksgränssnitt* och som svarar mot enheter som *eth0* eller *ppp0*, döljer sig i själva verket he-

[32]Vi har redan tidigare noterat att dessa gränssnitt inte renderar någon enhetsfil i */dev*. Nätverket hanteras inte som en block- eller teckenenhetsfil.

la TCP/IP-stacken från figur 9.1 i föregående avsnitt. TCP/IP är en naturlig del av Linuxkärnan och ligger därför automatiskt inkompilerad, så länge som någon nätverksdel valts under kompileringen av kärnan, d.v.s. i praktiken så gott som alltid.

9.3.1 Net-tools

Ett antal nödvändiga verktyg för att ställa in nätverket i GNU/Linux finns i programpaketet *Net-tools*. Precis som med t.ex. *Modutils* ligger detta programpaket utanför själva Linuxkärnan, och använder bara kärnan via de definierade gränssnitten.

Ett annat programpaket med namnet *Inetutils* innehåller de mest basala Internetverktygen i form av klient- och serverprogram och kommer att behandlas längre fram.

Varje distribution har oftast en uppsättning skript som ser till att konfigurera nätverkskorten, eller ringa upp en internetleverantör automatiskt under uppstart. (Uppringning kan också ske på begäran.) Dessa skript använder i sin tur i allmänhet endast programmen från paketet Net-tools, men ingår inte själva i Net-tools-paketet, utan hör till distributionen.

Det existerar ingen som helst standard för vart de mest grundläggande nätverksinställningarna skall lagras i ett GNU/Linux-system, utan detta är helt beroende på distribution. Red Hat / Fedora Core lagrar dem t.ex. i */etc/sysconfig/networking*, och Debian GNU/Linux lagrar dem i */etc/network*. För att få reda på hur det fungerar i just ditt system får du konsultera dokumentationen från din distributör eller själv rota runt i */etc* och *man*-sidorna.

Från dessa konfigurationsfiler hämtas vid uppstart av systemet en del information, som sedan används för att med hjälp av de olika verktygen i *Net-tools*-paketet konfigurera och starta nätverket. Om du bygger ditt eget system från grunden får du själv utveckla rutiner för hur nätverksinställningarna skall skötas, så du *kan* göra allt för hand om du vill. De verktyg och filer från paketet *Net-tools* som behövs för att ställa in nätverket för hand kommer nu att beskrivas.

Vissa av dessa kommandon är bara tillgängliga för root-användaren, men kan oftast köras även av vanliga användare som vill ha information om systemet, bara de skriver ut en fullständig sökväg, t.ex. **/sbin/ifconfig**.

arp utan argument ger information om vilka IP-adresser eller datornamn (t.ex. 192.168.1.64 eller foo.bar.se) som är kopplade till vilken MAC-adress (t.ex. 00:40:06:5D:63:CE) och vilket av datorns nätverksgränssnitt (t.ex. eth0) som en viss dator sist sågs till på.

Som syns av namnet är detta information som genereras och används av internetprotokollet ARP[33], och kallas följdaktligen för *ARP-tabellen* (engelska: *ARP cache*).

Den tabell som ges av kommandot skall utläsas som följer: om jag vill ha tag på datorn x, vilken MAC-adress har i så fall dess nätverkskort, och vilket av mina nätverksgränssnitt skall jag använda för att prata med det?[34]

De MAC-adresser som finns i ARP-tabellen avser *adresser på ditt lokala nätverk*. Om systemet vill skicka ett paket till en dator som ligger utanför ditt eget nätverk, kommer den att skicka det till en gateway istället, och då räcker det med att din gateway:s hårdvaruadress är känd. Att skicka paketet vidare på andra nätverk efter detta är inte ditt operativsystems huvudvärk, utan sköts av andra datorer. (Detsamma gäller när ett paket kommer till dig från ett yttre nätverk — det ser rent fysiskt ut att komma från en gateway.) ARP-tabellen på en hemdator innehåller oftast ganska få IP-nummer, men måste åtminstone innehålla gateway-datorns nummer för att kunna kommunicera med Internet.

När datorn startas upp är ARP-tabellen tom. När din dator vill nå en annan dator på nätverket kommer den att fråga efter deras hårdvaruadresser med broadcast, och därefter lägga till dem i ARP-tabellen. Det är möjligt att använda **arp** interaktivt för att lägga till eller ta bort datorer ur listan, men det får betraktas som mycket udda. Ett möjligt användningsfall är om du har flera nätverk anslutna till din dator och flyttar en annan dator från ett av nätverken till ett annat. Din dator kommer då att ha fel gränssnittsinformation för den flyttade datorn, och du kan uppdatera den för hand med **arp**.[35]

Kommandot **arp** är naturligtvis meningslöst för t.ex. PPP-anslutningar, eftersom en sådan anslutning inte behöver göra uppslag mellan IP-nummer och fysiska adresser på detta vis. Det finns bara en plats dit en dator med PPP-anslutning kan önska skicka ett paket, och det är till den andra punkten av förbindelsen.

hostname används för att ta reda på vad datorn heter i nätverket. Det går också att ange vad den skall heta genom att ge en parameter, t.ex. **hostname foo**. Denna förändring är dock högst tillfällig och

[33]Se avsnitt 9.2.7 på sidan 309.

[34]Om du bara har ett nätverkskort i din dator är detta såklart föga meningsfull information, men Linuxkärnan är gjord för att hantera många nätverkskort i en och samma dator.

[35]Enklast är kanske att starta om datorn istället, eller skriva **init 1** följt av **init 5**.

lär försvinna nästa gång du startar om datorn, och skall därför inte ändras på detta vis utan i den inställningsfil som används när datorn startas upp. (Ibland heter den filen */etc/hostname*, och ibland något helt annat.)

Några andra verktyg som i själva verket bara är symboliska länkar till **hostname** är:

dnsdomainname ger DNS-domännamnet för datorn. Du kan inte ändra domännamnet med detta kommando, och det vill du förmodligen inte heller göra, eftersom det bestäms av din DNS-server. Vill du se det fullständigt kvalificerade domännamnet (*www.foo.org*) kan du skriva **hostname --fqdn**.

domainname ger NIS- eller YP-namnet för denna dator. Se avsnitt B.1 på sidan 418 för mer information om dessa saker. Du kan ändra NIS/YP-domän med kommandot om du vill, men bara temporärt. Kommandona **ypdomainname** och **nisdomainname** har exakt samma funktion.

ifconfig (som skall utläsas *network interface configurator*, nätverksgränssnittskonfiguratorn) är den huvudsakliga arbetshästen för att konfigurera nätverket. Distributionernas uppstartskript kör detta program automatiskt vid uppstart med en rad parametrar för att ställa in de viktigaste nätverksinställningarna.

ifconfig -a ger en lista över installerade nätverksgränssnitt och alla deras nuvarande inställningar.[36] Detta är ett bra sätt att t.ex. ta reda på MAC-adressen till ett nätverkskort i datorn om du behöver det. Här kan du också få en överblicksbild över alla saker du kan konfigurera för ett nätverkskort.

Alla former av gränssnitt kan listas med **ifconfig**, men inte alla kan konfigureras och startas med det: PPP-gränssnittet är ett sådant. PPP-anslutning beskrivs därför i ett separat avnitt på sidan 321. Om du har ett Ethernetkort (vilket väl också är det vanligaste) kan du dock använda **ifconfig** direkt. För de allra flesta anslutningar duger:

```
# ifconfig eth0 192.168.1.64 netmask 255.255.255.0 up
# route add -net 192.168.1.1 netmask 255.255.255.0 eth0
```

Detta kommer att sätta upp en nätverksförbindelse med IP på en dator. *192.168.1.64* är ditt tilltänkta IP-nummer, och *255.255.255.0*

[36] Om du bara skriver **ifconfig** kommer alla *aktiva* anslutningar att listas.

din nätmask, samt *192.168.1.1* din standardgateway. (**route**-kommandot kommer att behandlas snart).

Du kan ha flera olika IP-nummer kopplade till ett och samma nätverksgränssnitt. Detta brukar kallas *aliasing*.[37] Om du i exemplet ovan vill koppla gränssnittet *eth0* till ytterligare ett nummer kan du skriva:

```
# ifconfig eth0:1 192.168.1.65 netmask 255.255.255.0 up
```

nameif kan köras innan **ifconfig** har körts på ett nätverksgränssnitt med kommandot **up**. Det används för att utifrån nätverkskortets MAC-adress ge det ett speciellt namn, t.ex. *foo* istället för det vanliga numrerade namnet (t.ex. *eth0*, *eth1* etc.). Kommandot används för att du skall kunna vara säker på att ett visst nätverkskort alltid heter *foo* oavsett om du sätter in ytterligare ett kort i datorn. Namngivningen av *ethN*-enheterna kan komma att ändras av detta.

nameif används knappt alls, eftersom få har detta specifika behov att kunna byta kort i datorn. När det körs läser det in filen */etc/mactab* som innehåller en lista med MAC-adresser och motsvarande gränssnittsnamn på formen:

```
foo 00:61:57:62:4A:24
bar 00:71:51:11:22:34
...
```

Alla efterföljande kommandon mot detta gränssnitt, t.ex. **ifconfig** eller **route** måste sedan köras mot det nya namnet för att fungera.

route är det viktigaste kommandot efter **ifconfig**. **route** utan några argument ger en tabell över *paketrutter* (engelska: *routes*). Eftersom Internetrutter är den vanligaste typen av rutt, kommer dessa att visas.

När ett paket skall skickas med IP-protokollet testas möjliga rutter uppifrån och nedåt i tabellen: adressen som paketet skall till utsätts för en logisk AND med nätmasken för den första posten i tabellen, jämförs med destinationsnätet, och matchar de så skickas paketet till det nätverksgränssnitt som anges sist på raden.

[37] Från det engelska *alias* som betyder ungefär *pseudonym*. Gränssnittet heter i exemplet 192.168.1.64 men är även känt som 192.168.1.65. (Litteraturvetarna kanske föredrar att kalla det för *nom de plume*.)

Två rutter som så gott som alltid är definierade är till klass A-nätet 127.0.0.1 och sist i listan *standardrutten* (engelska: *default route*) som har nätmasken 0.0.0.0 och som tar hand om alla adresser som inte matchat någon av föregående poster. På det viset kan helt obekanta datorer på Internet alltid kontaktas.

Det viktigaste användningsområdet för **route** är att lägga till standardrutten, med t.ex.:

```
route add default gw 192.168.1.1
```

Vilket anger att gateway-datorn på ditt lokala nätverk med IP-numret 192.168.1.1 skall ta hand om alla paket som inte tas om hand av något annat lokalt nätverk.

När du bygger lokala nätverk kan du använda **route** på allehanda innovativa sätt för att skyffla data mellan olika delar av ditt nät. Du kan t.ex. lätt ha flera olika gateway-datorer i ditt nät för att skicka mellan olika delnät. Det är också möjligt att ha flera olika gateway-öppningar till Internet.

netstat ger information om alla nätverksanslutningar och paketrutter som finns definierade i systemet vid ett givet tillfälle. Det kan även visa status för anslutningar när Linuxkärnan används som brandvägg eller *nätverksöversättare* (engelska: *network translator*).

netstat utan parametrar eller **netstat --protocol=unix** kommer att ge en lista över nätverksanslutningar med *Unixdomänsocklar* (engelska: *unix domain sockets*).[38] Dessa går till den egna datorn och är kanske inte särskilt intressanta annat än för programmerare. Ett vanligt användningsområde för dessa socklar är interprocesskommunikation (IPC) och om du kör detta kommando kan du se att din dator pratar ganska mycket med sig själv.

netstat --protocol=inet är nog det vanligaste sättet att använda **netstat**. Detta ger en lista över aktiva anslutningar till och från den egna datorn, som använder internetprotokollet. (Ett alternativ som fordrar färre knapptryckningar är **netstat -Ainet**.) Detta ger en lista över protokoll (oftast TCP), hur mångar paket som ligger i kö på förbindelsen (oftast 0, om det är snabba fina anslutningar), lokal IP-adress eller datornamn och port, samt mottagande dators IP-adress och port.

[38] Det korrekta standardnamnet är dock POSIX Local IPC sockets, se även avsnitt 2.3.5 på sidan 73.

Den mottagande datorns port kan ibland anges direkt med protokollnamnet, t.ex. *foo.bar.org:ssh* om du har en SSH-session öppen mot den datorn. Därutöver anges även tillståndet för förbindelsen. Om du misstänker att någon är inne och rotar i ditt system och har tagit sig in via Internet, avslöjar detta kommando ofta allt.[39]

netstat -r ger samma resultat som kommandot **route** utan argument: en lista över paketrutter.

plipconfig används för att konfigurera en IP-anslutning via datorns parallellport, så att två datorer kan utbyta data via en printerkabel. Detta kallas PLIP (Parallell Line Internet Protocol) och är väl inte speciellt användbart för en normal användare, men den som är innovativ kan ibland vilja ansluta en riktigt gammal dator utan Ethernetkort till Internet genom att utnyttja parallellporten på en annan dator. Ett vanligt användningsområde är att ansluta nätverk till gamla bärbara datorer utan inbyggt nätverkskort.

rarp används för att ta reda på en dators IP-nummer om du vet dess MAC-adress. Detta används nästan inte alls, och de flesta förkompilerade Linuxkärnor saknar också stöd för detta. Ibland är själva programmet t.o.m. borttaget ur distributionen.

slattach används för att ansluta en annan dator med SLIP, föregångaren till PPP. Eftersom PPP är det dominerande systemet för seriekabelanslutningar nuförtiden har detta förmodligen spelat ut sin roll. (PPP kommer att behandlas i nästa avsnitt.)

Eftersom det är ganska jobbigt att konfigurera en dators nätverk med **ifconfig** och **route** medföljer det ibland ett par bonusskript i vissa distributioner:

ifup är det icke-standardiserade program som din distribution i slutänden brukar köra för att starta varje slag av nätverksförbindelse. Programmet ligger ofta i */sbin*-katalogen i ditt system. I en del system är detta ett kompilerat, binärt program, och i andra ett skript. Det läser in inställningar från någon/några filer i */etc* och startar sedan nätverket med hjälp av **ifconfig** och **route**. Det körs oftast med **ifup eth0**. Om du använder DHCP kommer oftast **dhclient** (se längre fram i detta kapitel) att anropas automatiskt.

[39]Om inte inkräktaren har varit listig nog att byta ut själva kommandot **netstat** naturligtvis, vilket är nog så vanligt. Du kan läsa om detta i avsnitt 10.5.1 på sidan 359.

ifdown deaktiverar ett aktivt nätverksgränssnitt. Typiskt användningsfall: **ifdown eth0**. Distributionerna brukar köra detta program när datorn stängs av.

9.3.2 DNS-konfiguration

Utöver konfigurationen med **ifconfig** och **route** måste du även ha tillgång till en DNS-server för att kunna använda vanliga symboliska datornamn (t.ex. *www.foo.org*) för att anropa andra datorer på Internet.

Detta kallas med en svengelsk term för *namnupplösning* efter engelskans *name resolution*. Ett separat programbibliotek knutet till Linuxkärnan har hand om denna del av operativsystemet.

Namnet på en eller flera DNS-servrar skall alltid, om du inte använder DHCP eller PPP (som ju automatiskt tillhandahåller DNS-servrar), anges i filen */etc/resolv.conf*. Detta varierar inte mellan distributioner som använder GNU C Library[40] och filen innehåller oftast bara två-tre rader i stil med:

```
domain bar.org
nameserver 192.168.1.2
```

Den första raden anger ett domännamn, så om din dator heter *foo* skall dess fullständiga namn vara *foo.bar.org*, och namnet *foo* skall finnas i */etc/hostname* eller motsvarande fil. Detta innebär dock *inte* att din dator har registreras i DNS-servern så att andra kan hitta din dator med hjälp av detta namn; en sådan registrering kan bara en DNS-serveradministratör göra, och registrering av hela *domäner* sköts av ett fåtal landsomfattande organisationer.

Det huvudsakliga användningsområdet för *domain*-parametern är om du t.ex. anger ett visst datornamn utan närmare specifikation, t.ex. **ping fnord**. Då kommer kärnan att anta att det är datorn *fnord.bar.org* som avses.

Den andra raden anger en DNS-server, som inte behöver vara placerad på det lokala nätverket. */etc/resolv.conf* kommer att laddas av kärnan när det första programmet som behöver DNS-uppslagning körs. Om du har tillgång till flera olika DNS-servrar kan flera IP-nummer anges här.

I filen */etc/hosts.conf* anges vilka metoder namnupplösarbiblioteket skall använda för att översätta ett givet datornamn till ett IP-nummer. Det normala är att filen bara innehåller raden:

```
order hosts,bind
```

[40] I princip alla program för GNU/Linux använder GNU C Library för att göra namnupplösning.

Detta talar om för namnupplösaren att den först skall försöka slå upp datornamnet i filen */etc/hosts* och därefter, om detta misslyckas, försöka med den DNS-server som angivits i */etc/resolv.conf*.

Om du har en instabil DNS eller manuellt vill ändra namnuppslagningen kan du modifiera filen */etc/hosts*. Denna fil innehåller normalt bara en rad som kopplar datornamnet *localhost* (i olika varianter) till den egna datorns IP-nummer 127.0.0.1.

Vill du blockera tillgången till en viss dator kan du t.ex. skriva in *127.0.0.1 www.fnord.com* i */etc/hosts* för att skicka alla försök att ansluta till *www.fnord.com* till den egna datorn. Detta blockerar dock ingen användare som är listig nog att använda IP-adressen direkt.

Kommandot **host** kan användas för att manuellt slå upp IP-numret för en viss dator, så om någon systemadminstratör blockerat tillgången till datorn *www.fnord.com* för dig kan du ta reda på den datorns IP-adress med **host www.fnord.com** och sedan surfa till IP-numret som anges. Kommandot kan också användas på omvänt vis i de fall du bara vet ett IP-nummer och vill veta vilket datornamn det IP-numret svarar mot. (Kommandot **host** hette tidigare **nslookup**.)

Tycker du att **host** inte ger tillräckligt mycket information finns även kommandot **dig** att tillgå. Detta fungerar på samma vis, men ger även mängder med DNS-protokollrelaterad information till den intresserade.

Kommandot **whois** är tänkt att användas för att ta reda på vem som ligger bakom en viss domän. **whois foo.org** skall till exempel ge namn, adress och telefonnummer till innehavaren av domänen *foo.org*. De svenska domänerna anses dock inte kunnas lämnas ut på detta vis p.g.a. viss lagstiftning, och dessa måste därför istället undersökas via ett webbgränssnitt.[41]

Det finns ytterligare en form av namnupplösning, som innebär att du använder katalogtjänster (engelska: name service) som t.ex. NIS. Detta kommer inte närmare att behandlas här, men kan konfigureras i filen */etc/nsswitch.conf*. Läs mera om detta i avsnitt B.1 på sidan 418.

9.3.3 DHCP med dhclient

Om du nu har tillgång till en DHCP-server att hämta inställningar till ditt nätverkskort ifrån blir det hela mycket enklare. I princip alla inställningar vi gått igenom ovan kan skötas med hjälp av DHCP, så som tidigare nämnts. Till detta används uteslutande programmet **dhclient** som distribueras separat från Net-tools.

[41] Se: `http://www.nic.se/domaner/domansok.shtml`

I de flesta fall används **dhclient** av skriptet **ifup** eller någon liknande komponent i en distribution. Det går dock utmärkt att köra för hand också, t.ex.:

```
# dhclient eth0
```

Kommandot måste utföras av root, och får bland annat till följd att innehållet i filen */etc/resolv.conf* automatiskt ändras, ditt nätverksgränssnitt skall också ha fått ett IP-nummer och en nätmask som du kan se med **ifconfig** och en standardrutt som du kan se genom att skriva **route**.

Om du är nyfiken på alla inställningar som **dhclient** har fått från DHCP-servern kan du undersöka filerna i */var/lib/dhcp*. Där lagras DHCP-informationen mellan anropen till servern. Vissa DHCP-servrar skickar ut mer information än andra.

9.3.4 PPP-anslutning

Modemanslutningar med PPP konfigureras genom att ett separat program eller s.k. ”chat-skript”, i praktiken oftast ett program med namnet **wvdial** (namngett efter företaget Worldvisions som skapade programmet), ringer upp en modempool via ett modem anslutet till en av serieportarna (ttyS0, ttyS1 etc.). Själva uppringningen sker med modemkommandon, vilket vanligen är s.k. Hayes-kommandon.[42] Ett Hayeskommando kan du själv skicka till ditt modem med t.ex. **echo ”atz” > /dev/ttyS0** (om ditt modem råkar sitta på den porten). Vanligtvis behöver du inte befatta dig med dessa kommandon, eftersom uppringningsprogrammet sköter det åt dig.

Efter uppringningen kan skriptet ibland behöva skicka kommandon till servern på andra sidan, men oftast inte. När själva uppringningen är klar och modemet anslutet till PPP-servern lämnar chat-skriptet eller **wvdial** över kontrollen till en demon med namnet **pppd**. Denna demon som har skrivits av Paul Mackerras kan även användas med bl.a. Solaris.

pppd-demonen tar över den serieport som det föregående programmet anslutit sig till servern på, och börjar då använda själva PPP-protokollet: den förhandlar med servern om användarnamn och lösenord (via LCP), IP-adress, nätmask och DNS-server (via IPCP) och sätter till sist upp förbindelsen för nätverksgränssnittet *ppp0* (eller högre gränssnittsnummer, eller *ippp0* för ISDN, o.s.v.).

Att avsluta förbindelsen är även det distributionsberoende, men **wvdial** kan startas i ett eget terminalfönster, och väntar då på att du skall

[42] Dessa kommandon har fått sitt namn efter modemtillverkaren Hayes Communications, som i sin tur fått namnet efter grundaren Dennis Hayes.

avsluta processen genom att trycka **Ctrl+c** i detta fönster. Förbindelsen kommer då att kopplas ned.

För att konfigurera datorn att använda PPP över modem beror tillvägagångssättet helt på din distribution, och ofta finns det helt grafiska hjälpverktyg som sköter detta. För en "ren" dator utan grafisk användarmiljö o.dyl. kan du installera **wvdial** och **pppd**, och som root köra **wvdialconf /etc/wvdial.conf**. Se till att ha modemet anslutet och påslaget, så kommer konfigureringsprogrammet automatiskt att upptäcka modemet och konfigurera det. Du måste sedan ange telefonnummer, användarnamn och lösenord till modempoolen i den skapade filen */etc/wvdial.conf*.

Därefter kan PPP konfigureras ytterligare i ett antal filer i katalogen */etc/ppp*. Det behövs normalt inte.

Som tidigare sades kan du sedan starta din konfigurerade förbindelse med kommandot **wvdial** i ett terminalfönster, och avsluta med **Ctrl+c** när du inte vill vara ansluten längre.

9.3.5 ADSL-anslutning

ADSL (*Asymmetric Digital Subscriber Line*) är en standard för att åstadkomma höghastighets dataöverföring på en vanlig telefontråd av koppar. På telenätet används vanligen modem till detta, men om teletrafiken kopplas från helt och utrustningen i båda ändar dynamiskt kan anpassas till trådens signalöverföringskarakteristika, kan linjen utnyttjas mer effektivt för dataöverföring.

Den låda som utgör abonnentdelen ansluts i abonnentens bostad och kallas ibland lite slarvigt för *kabelmodem*. Den andra delen av utrustningen ansluts på telestationen. I övrigt är ADSL-anslutningen ett helt vanligt Ethernetuttag och kräver ett nätverkskort av Ethernet-typ för att anslutas till datorn.

De flesta svenska internetoperatörer som levererar ADSL-anslutningar till Internet använder ett DHCP-baserat system med tillägget att inloggning måste göras via en webbsida. System av detta slag är vanligt i trådlösa nätverk, eftersom vem som helst lätt kan ansluta sin utrustning till nätet, och separat inloggning krävs då för att avgöra om den som anslutit sig verkligen har rätt att använda nätet.

Detta är ingen som helst Internetstandard utan baseras på *Orbyte Wireless System* från företaget Åkerströms Nowire AB. Syftet med den extra inloggningen är sannolikt att förenkla rutiner och bokföring: den centrala servern håller reda på vilka abonnemang som är aktiverade. Till de operatörer som använder eller har använt detta system hör Telia, COMHEM och Tiscali.

Andra internetleverantörer nöjer sig med att t.ex. mekaniskt ansluta och koppla från kabeluttaget i en viss lägenhet, eller registrerar nätverkskortens MAC-adresser i en DHCP-server för att avgöra om en viss dator skall släppas in i nätet eller ej. Denna typ av administration blir allt vanligare och kommer på sikt säkert att ersätta lösningar med inloggning.

Att fylla i användarnamn och lösenord varje gång en uppkoppling skall göras är inte speciellt kul i längden, så för att underlätta användandet av detta system under Linux skapade Jakob Stasilowicz det fria programmet *GNU Alive* (tidigare *qADSL*).[43] Eftersom detta system är väldigt specifikt för Sverige ingår det i få distributioner, och du måste därför installera det själv. Det finns dock med i Debian GNU/Linux.

GNU Alive är en demon som automatiskt loggar in dig hos en Internetleverantör som använder Orbyte Wireless System, och därefter håller förbindelsen levande. Manualsidorna för GNU Alive innehåller ytterligare information om hur förbindelsen konfigureras och sätts upp.

9.3.6 Trådlösa nät

Ett Linuxsystem kan naturligtvis även använda trådlösa nätverk, på engelska kallade *Wireless LAN* eller kort och gott bara *WLAN*. Standarden dessa kort grundar sig på kallas *Wi-Fi* och i denna ingår standarderna i 802.11-gruppen.

Trådlösa nätverk identifierar sig med ett ESSID, *Extended Service Set Identifier*, vilket i praktiken bara är ett enkelt kortnamn, en sträng. Eftersom ett och samma nätverk kan ha flera sändare som är tillgängliga inom samma område och eftersom det också kan finnas flera nätverk på samma plats, måste nätverkskortet veta vilka sändare som hör till vilket nätverk genom att undersöka deras ESSID. Om t.ex. bolaget *Foo* har fyra sändare på en viss plats och bolaget *Bar* har fem, kommer du att kunna veta vilka som hör till just din operatör genom att operatör Foo:s fyra sändare har ESSID ”foo” medan operatör Bar:s sändare använder ESSIS ”bar”. Om du anger att du vill använda ESSID ”foo” kommer nätverkskortet vid behov att kunna växla mellan Foo:s fyra olika sändare utan problem.

Ett trådlöst gränssnitt i Linux fungerar precis som ett vanligt nätverksgränssnitt med den enda skillnaden att det eventuellt har ett fantasifullt namn, vanligtvis *wlan0*. För att få igång och använda detta gränssnitt behöver du dock två saker:

- En drivrutin till Linuxkärnan som stödjer det trådlösa nätverkskort du har tänkt använda.

[43] Se: `http://www.gnu.org/software/alive/`

- Programpaketet *Wireless-tools*.

Det första är oftast det mest problematiska. Trådlösa nätverkskort lägger stora delar av ansvaret på operativsystemet, och det är inte vanligt att drivrutiner för Linux har utvecklats vare sig som del av kärnan eller separat. Det är därför ofta en god idé att undersöka i förväg om ett visst kort stöds av Linux eller inte. Var mycket noga med produktnamnet: det är inte ovanligt att en tillverkare byter *chipset* (de kretsar som sitter i själva kortet) mellan två minimala uppdateringar av själva produkten. Detta innebär att produkten ser helt annorlunda ut för operativsystemet och ofta kräver en helt annan drivrutin.

De flesta WLAN-kort brukar vara av PCMCIA-typ (senare kallat *PC Card* eller *CardBus*. För Linuxkärnans del är detta oviktigt; de modernare korten kommer att behandlas som vilket PCI-kort som helst, och du kan se information om dina PCMCIA-kort med kommandot **lspci** precis som för den mesta annan hårdvara som sitter i din dator. Med **lspci -v** kan du ofta också se vilket chipset som används av det trådlösa nätverkskort som sitter i din dator.

När du väl lokaliserat det chipset och den drivrutin som fordras för ett visst trådlöst nätverkskort får du ofta kompilera om Linuxkärnan eller i bästa fall en enskild modul för att kunna få kortet att fungera. Det behövs ofta dessutom en s.k. *firmware*, vilket kort uttryckt är ett datorprogram som skall köras i nätverkskortets processor. Vissa kort har till och med flera processorer, och behöver därför flera olika firmwarefiler. Att få tag på dessa drivrutiner och filer är inte alltid enkelt, och det enklaste är oftast att följa instruktionerna som finns på webbsidorna för de olika drivrutinsprojekten. Dessa projekt förutsätter ofta att du har installerat Wireless-tools och är förtrogen med detta. Wireless-tools innehåller bland annat följande viktiga verktyg:

iwconfig är Wireless-tools motsvarighet till Network-tools **ifconfig** och är en universella Schweiziska armékniven för att konfigurera ett trådlöst nätverkskort. De vanligaste kommandona är dessa:

  iwconfig utan parametrar ger en lista över alla nätverksgränssnitt. För de som är trådlösa kommer ytterligare information att visas, bland annat vilken Wi-Fi typ kortet stödjer, vilket ESSID kortet försöker ansluta sig till, och vilken radiokanal det använder. Där finns även mer detaljerad information om kryptering och radiosändarens kapacitet med mera.

  iwconfig wlan0 essid "foo" kommer att ange att detta nätverksgränssnitt skall använda ESSID:t "foo". Detta måste nästan alltid anges. Om du sätter upp ett eget trådlöst nätverk hemma är det möjligt att du anger ESSID:t "" (tom sträng) men

om dina grannar börjar göra samma sak är det inte särskilt lyckat. Om du kör detta kommando får det till följd att kortet försöker ansluta sig till det angivna nätet.

iwconfig wlan0 channel 11 talar om att nätverkskortet alltid skall använda kanal 11. Eftersom kanal 1, 6 och 11 alltid brukar finnas tillgängliga och inte överlappar andra kanaler kan stabiliteten i ett nätverk öka om du gör denna inställning, men normalt skall det inte behövas.

iwconfig wlan0 mode Managed anger att drivrutinen själv skall byta mellan sändare och kanaler vid behov. Det finns tillfällen, om du t.ex. vill koppla samman två datorer, då läget *Adhoc* används istället.

Att bara köra dessa **iwconfig**-kommandon (oftast behövs flera av dem) tar dig så långt att du anslutit kortet till en viss sändare för ett visst nät, och en viss kanal. Därefter fordras den vanliga nätverkskonfigurationen med **ifconfig**, men eftersom nästan alla trådlösa nät använder DHCP räcker det oftast att skriva **dhclient wlan0** efter att radioinställningarna är klara. De distributioner som stödjer trådlösa nätverkskort har naturligtvis automatiska konfigureringsprogram för alla dessa uppgifter.

Andra användningsområden för **iwconfig** är att ställa in krypteringsnycklar för att kryptera trafiken mellan nätverkskortet och radiobasstationen och diverse andra komplicerade saker. Dessa kan med fördel läsas på manualsidan för verktyget.

iwlist är ett verktyg som tar hand om det mesta som **iwconfig** inte klarar av. Det allra vanligaste är att lista alla tillgängliga nätverk med t.ex. **iwlist wlan0 scan**. Om du inte anger ett nätverksgränssnitt kommer **iwlist** att försöka använda alla nätverksgränssnitt som stödjer trådlösa nät för att lista tillgängliga nätverk. Det kan ofta bli en ganska lång lista. I princip alla nätverk du hittar vid en sådan listning brukar det vara möjligt att ansluta sig till, men det innebär inte att du med automatik kan använda dem (se nedan).

Även om du ofta lyckas med att ansluta till ett trådlöst nätverk med hjälp av dessa verktyg och DHCP, så att din dator har alla nödvändiga nätverksinställningar, betyder inte detta automatiskt att du har tillgång till Internet. Ofta behövs en inloggning via en webbsida som ”öppnar upp” din gateway och släpper fram trafiken till Internet. Detta krävs av naturliga skäl: de flesta trådlösa nät är kommersiella (t.ex. Telia HomeRun) och kräver att du betalar för att få tillgång till uppkoppling-

en. Många hotell och liknande kräver också inloggning eftersom de vill hindra personer i angränsande lokaler från att använda nätverket.

9.3.7 Inetutils

Inetutils är en uppsättning standardprogram som skapats för internetprotokollen av GNU-projektet. Flera av dem är enkla kommandoradsverktyg, medan vissa är demonprogram som skall köras i bakgrunden för att en dator skall kunna agera server för en viss tjänst. En delmängd av Inetutils följer så gott som alltid med varje GNU/Linux-distribution, men vissa delar kan vara utelämnade eftersom de anses osäkra eller urmodiga.

ftp implementerar klientdelen av FTP-protokollet och gör det möjligt att hämta filer med TCP som underliggande transportprotokoll. Det går bra att ansluta till en FTP-server på Internet med kommandoradsverktyget bara genom att skriva **ftp foo.bar.org** och klienten kommer då att ansluta till denna dator som i sin tur begär lösenord. Programmet används i princip bara av gamla stofiler som jag själv, eller av skript. Anledningarna är två:

För det första är FTP så osäkert att det inte går att använda för att överföra filer mellan två datorer där du har konto. Lösenordet skickas i klartext över Internet och kan lätt avlyssnas av vem som helst som har tillgång till den fysiska kabeln. Därför används nästan uteslutande **scp** eller **sftp** (se avsnitt 9.5) till filöverföring mellan konton numera. Det enda undantaget är då filer hämtas anonymt, men även i detta fall ersätts FTP numera ofta med protokollet HTTP.

För det andra är det få som använder FTP från kommandolinjen, de flesta använder något grafiskt FTP-program, t.ex. **gftp**, eller skriver en FTP-URL i sin webbläsare, i stil med *ftp://foo.bar.org/*. En sådan URL kommer automatiskt att logga in på FTP-servern som användaren *anonymous*, men ett konto kan också anges med *ftp://user:pass@foo.bar.org/*.

För att använda FTP för att logga in anonymt på en FTP-server anger du användarnamnet *anonymous* följt av en epostadress istället för lösenord.

När du väl är ansluten till en FTP-server kan du använda olika kommandon från FTP-standarden för att lista filer på servern och eventuellt ladda ner några av dem, eller ladda upp någon fil. Observera att dessa kommandon *inte* är POSIX-kommandon eller

något annat utan enbart FTP-kommandon som skickas till FTP-servern: den förvillande likheten med existerande kommandon i POSIX är bara avsedd att förenkla användandet.

ls listar filerna som finns lagrade i den aktuella katalogen på FTP-servern. Även kommandot **dir** finns att tillgå.

cd foo går ned i katalogen *foo* på FTP-servern. Även andra intuitiva katalognavigeringsmöjligheter som t.ex. **cd ..** fungerar bra.

lcd foo går ned i katalogen *foo* på den egna datorn istället för på FTP-servern.

get foo.txt hämtar ("laddar ner") filen *foo.txt* till aktuell katalog på den egna datorn från FTP-servern.

put foo.txt sänder ("laddar upp") filen *foo.txt* till FTP-servern. Ofta kan du i fallet med anonym FTP bara göra detta till vissa kataloger med ledande namn som t.ex. *upload*.

mget foo* hämtar flera filer med ett vanligt globbningsuttryck[44] och frågar vid propmpten för varje fil om du verkligen vill ladda ned den. **mput foo*** ger samma funktion vid uppladdning.

Med i Inetutils följer också **ftpd** som gör det möjligt att köra en egen FTP-server i form av ett demonprogram.

ping är ett av nätverksadministratörens favoritprogram. Det används för att testa om en annan dator på nätverket är tillgänglig via internetprotokollet.

ping skrevs urspungligen av Michael Muuss och namnet kommer från ubåtssonartekniken, där ett *ping* skickas med hjälp av en sonar[45] för att mäta avståndet till föremål i närheten genom att se hur lång tid det tar för ekot att komma tillbaka. På samma vis ger **ping** en bedömning av närvaron och svarstiden för en viss dator genom att skicka ICMP-paket till den och invänta svar. Genom att skriva **ping foo.bar.org** kommer du att skicka ett antal ICMP-paket till den avsedda datorn, och du får sedan uppgift om hur lång tid det tar för varje ICMP-paket att färdas fram och tillbaka till datorn.

[44] Se avsnitt 2.3.6 på sidan 75.

[45] *Sonar* är en akronym för *sound navigation and ranging*. När det träffar ett fartygsskrov ger det upphov till det dova, obehagliga ljud som ofta förekommer i ubåtsfilmer som *Das Boot* (eller för den delen samplat i Antiloop-låten *Nowhere to Hide*).

Vad som är bra svarstider är en erfarenhetssak: *avstånd* på Internet mäts just i svarstid på ICMP-paket (d.v.s. **ping**) och har inget att göra med geografiskt avstånd, vilket *kan* vara relaterat, men inte behöver vara det. Avståndet mellan två punkter kan också variera något beroende på nätverkets belastning.

Genom att skriva t.ex. **ping -b 192.168.1.255** kan du skicka ett ping till alla datorer i det lokala nätverket, och du skall då få svar från samtliga startade datorer.

rsh (remote shell) liksom **rcp** (remote copy) och **rlogin** (remote login) är på samma vis som **ftp** gamla, osäkra protokoll för anslutning till en annan dator. Inget av dem skall användas! Använd istället SSH-svitens **ssh** (svarar mot **rsh** och **rlogin**) för att ansluta till en annan dator och utföra kommandon, samt **scp** (secure copy) för att kopiera filer mellan olika datorer och konton.

talk är ett system för att prata interaktivt med andra användare på datorer på Internet. Påminner till viss del om IRC (Internet Relay Chat) men är inte lika förgrenat. **talk** används mest för att tala med andra användare på en och samma dator, som är anslutna via andra terminaler.

telnet är en implementation av Telnet-protokollet som gör det möjligt att logga in på andra datorer på Internet. Det som gäller för **rlogin** och **rsh** gäller även **telnet**: *använd det inte* för fjärrinloggningar! Telnet är totalt osäkert eftersom lösenord och allt du skriver mycket lätt kan avlyssnas direkt på nätverkskabeln. För fjärrinloggning skall du använda **ssh** istället (se sidan 331). Telnet-demonen **telnetd** som också medföljer Inetutils och som möjliggör fjärrinloggningar med Telnet skall under inga omständigheter startas på ditt system!

Telnet är dock ett så simpelt protokoll att det kan användas till att ansluta till alla möjliga andra tjänster för att testa att de fungerar: det allra vanligaste tricket är att använda telnet för att testa att en webbserver svarar som den skall: skriv t.ex. **telnet www.dn.se 80** för att ansluta till port 80 på Dagens Nyheters webbserver, och skriv sedan **GET /** följt av **ENTER**. Du får då först en avdelning med HTTP-meddelanden, och därefter HTML-koden för Dagens Nyheters förstasida.

Liknande trick kan användas för att testa flera av de protokoll som återfinns i figur 9.2 och avslöjar lite om hur enkelt uppbyggda internetprotokollen på högre nivå faktiskt är: i grunden är de alla rena, enkla textmassor som ges som svar på kommandon skickade i

klartext, som om de skrivits av en människa i ett terminalfönster. Populärt numera är t.ex. "webbtjänster" vilket egentligen inte är något annat än textmassor formaterade enligt XML-standarden, och transporterade över HTTP via port 80 på exakt det vis som illustrerades ovan med Dagens Nyheters webbserver.

tftp är en enkelt filöverföringsverktyg baserat på TFTP-protokollet (RFC 1350), som använder UDP som transportprotokoll. Det är betydligt mindre begåvat än FTP, men används en del för att t.ex. överföra uppstartfiler till *X*-terminaler i en del nätverk. Det är naturligtvis också lika osäkert som FTP och skall inte användas för någon form av dagligt arbete. Inetutils innehåller även en server för TFTP med namnet **tftpd**.

9.4 Xinetd och TCPd

Officiell hemsida: `http://www.xinetd.org/`

En mängd möjliga demoner kan köras för att tillhandahålla tjänster på din dator. T.ex. körs **ftpd** för att erbjuda FTP och **tftpd** för att tillhandahålla TFTP, o.s.v.

Men det är inte alltid så att dessa demoner jämt och ständigt körs på din dator. **ftpd** finns t.ex. nästan aldrig med i processlistan om du skriver **ps -A**. Vissa demoner startas nämligen indirekt från en annan demon: **inetd** eller **xinetd**, vilket står för *Internet Services Daemon* respektive *Extended Internet Services Daemon*. Det är den senare som är i mer allmän användning numera, men det som står i detta avsnitt gäller även det äldre programmet, med vederbörliga modifikationer.

Vi vet att en demon kan ha som uppgift att svara på ett anrop till en viss TCP- eller UDP-port på datorn. Det betyder, att när ingen anropar datorn för att få tillgång till en viss tjänst, ligger demonen bara och sover och tar upp minne. För att du skall slippa ha onödigt många demoner startade, som bara har till uppgift att svara på enstaka anrop då och då, väljer många att låta demonen **xinetd** lyssna på ett antal portar och starta demoner då ett anrop kommer in, istället för att ständigt ha demonerna körande.

Om **xinetd** är installerat i ditt system använder den i sin tur en annan demon, **tcpd** för att avgöra om ett anslutningsförsök utifrån skall släppas in eller ej. Demonen **tcpd** kommer då i sin tur att använda filerna */etc/hosts.allow* och */etc/hosts.deny* för att avgöra om anslutningen

är tillåten. Dessa filer kan styra tillgången genom att ange ett eller flera domännamn eller IP-nummer.[46]

I filen */etc/xinetd.conf* kan du konfigurera vilka demoner **xinetd** skall starta vid ett anrop till datorn, och därmed också vilka portar den skall lyssna på. Ibland använder distributionerna en hel katalog, */etc/xinetd.d* för konfigurationen, där varje fil representerar ett program som skall kunna startas.

Anledningen till att inte alla inställningar kan göras i */etc/xinetd.conf* är att program som installeras efter hand måste kunna anmäla sitt intresse av att få bli startade av **xinetd**. Installationsprogram för programpaket är dåliga på att editera konfigurationsfiler, men bra på att bara installera en ny fil på en viss plats, och därför kan de lätt berätta för ditt system att de vill bli startade genom att placera en konfigurationsfil i */etc/xinetd.d*.

Du bör alltid titta över filen */etc/xinetd.conf* och katalogen */etc/xinetd.d* i ditt system för att kontrollera att du inte har en massa onödiga tjänster startade. Om du upptäcker att du inte har nytta av någon enda av de saker som **xinetd** startar, så kan du naturligtvis stänga av den helt genom att modifiera dina uppstartskript. Att ha onödiga tjänster startade är en säkerhetsrisk.

Det är också möjligt att i den/dessa filer konfigurera tillgång *per tjänst*, d.v.s. filerna */etc/hosts.allow* och */etc/hosts.deny* avgör vilka tjänster som skall få komma in över huvud taget, medan **xinetd** i sin tur kan bestämma att bara *en* specifik dator får tillgång till *en* specifik tjänst.

För ett hemmasystem är kanske den huvudsakliga uppgiften för **xinetd** att hålla igång skrivarsystemet CUPS (se sidan 267), och det används också för att starta t.ex. Samba (se sidan 413).

Fördelen med att använda **xinetd** för att starta olika demoner är att **xinetd** kan sätta begränsningar på hur många demoner som får startas av olika slag, så att systemet inte blir överbelastat, samt att åtkomstkontroll till olika tjänster kommer gratis. Trots detta använder många serverdemoner inte **xinetd**: webbservern Apache och SSH-servern **sshd** är exempel på sådana demoner. Anledningarna till detta varierar: vissa anser att **xinetd** är värdelöst och använder det inte alls, andra använder det till allt.

[46] De flesta väljer dock att göra sådana begränsningar med hjälp av *Netfilter*, se avsnitt 10.6 på sidan 360.

9.5 SSH

SSH, som utläses *secure shell* är dels ett mer eller mindre standardiserat internetprotokoll, dels namnet på ett program som används för att utnyttja detta protokoll. Protokollet SSH används för att göra en krypterad[47] fjärranslutning från en dator till en annan via ett IP-nätverk, t.ex. Internet, och öppna ett skal på den mottagande datorn. SSH kan också verifiera att datorn som anropas verkligen är den dator den utger sig för att vara.

SSH var ursprungligen ett program som skrevs av Tatu Ylönen i Finland år 1995 och som distribuerades som fri mjukvara. Eftersom intresset för SSH var stort valde Ylönen att exploatera uppfinningen genom att starta ett företag med namnet SSH Communications Security Oy, som sedan dess sålt nyare versioner av programmet via företaget F-Secure.

Problemet med att ingen modern, fri version av SSH fanns allmänt tillgänglig åtgärdades år 1999 då svensken Björn Grönvall tog den ursprungliga fria versionen av SSH och rättade en del fel, varefter han kallade resultatet OSSH. I detta läge uppmärksammades OSSH av OpenBSD-projektet, som med sitt höga fokus på säkerhet ville ha ett säkert och fritt fjärrinloggningsprogram. På kort tid vidareutvecklade och sjösatte en grupp medlemmar från OpenBSD-projektet en fullständig version av SSH under namnet OpenSSH, med stöd för både version 1 och version 2 av SSH-protokollet. Snart därefter anpassades OpenSSH så att det även gick att köra under GNU/Linux.

GNU-projektet har en egen implementation av SSH-protokollet som utvecklas av svensken Niels Möller under namnet **lsh**. Denna version stödjer bara version 2 av SSH-protokollet.

IETF arbetar med att göra SSH-protokollet till en internetstandard, och har en arbetsgrupp som jobbar med standarddokumenten, men har ännu inte publicerat någon specifikation.

70% av de SSH-demoner som hittas på Internet använder i dagsläget OpenSSH. På grund av problem med exportrestriktioner rörande kryptoprodukter har alla kryptografiska delar av SSH utvecklats utanför USA. Den syntax som anges i följande stycken har testats för att fungera med OpenSSH. Fyra huvudsakliga program ingår i de flesta SSH-varianter:

ssh är den mest grundläggande komponenten och handhar fjärrinloggningar. Du loggar in på en annan dator med:

[47] Har du ingen aning om vad kryptering är, så smygläs avsnitt 10.4 på sidan 352.

```
# ssh fnord@foo.bar.org
```

Detta betyder att du vill logga in på datorn med namnet *foo.bar.org* med användarnamnet *fnord*. Om du inte anger något användarnamn kommer OpenSSH att anta att du heter samma sak på det andra systemet som du heter på den dator där du redan befinner dig, och kommer då att använda det det aktuella inloggningsnamnet. Den mottagande datorn kommer att fråga dig efter ett lösenord, och kommer därefter att öppna ett skal där du kan använda alla kommandon och starta program precis som vanligt. Det går naturligtvis bra att ansluta från/till helt andra operativsystem än det du själv använder.[48]

scp utläses *secure copy* och är ett program för att kopiera enstaka filer. Det används på liknande vis som **ssh**, men som hos de flesta POSIX-kommandon måste både källa och destination anges. Om jag t.ex. är inloggad på en viss dator, och har en fil med namnet *foo.txt* som jag vill kopiera till datorn *foo.bar.org*, där jag har ett användarkonto med namnet *fnord*, skriver jag:

```
# scp foo.txt fnord@foo.bar.org:
```

Lägg märke till det avslutande kolonet: efter detta följer nämligen det filnamn som du vill att filen skall ha på den mottagande datorn. Eftersom jag vill ha samma filnamn på den andra datorn som på datorn jag redan befinner mig på, skriver jag inget efter kolonet. Filken kommer då att lagras i min hemkatalog på den mottagande datorn, under namnet *foo.txt*. Om jag däremot vill att filen skall lagras under namnet */tmp/bar.txt* på den mottagande datorn skriver jag istället:

```
# scp foo.txt fnord@foo.bar.org:/tmp/bar.txt
```

Om jag istället vill *hämta* filen foo.txt från min hemkatalog på den andra datorn, skriver jag:

```
# scp fnord@foo.bar.org:foo.txt .
```

[48]SSH-klienten PuTTY som finns till alla versioner av Microsoft Windows är t.ex. mycket populär bland användare av olika POSIX-system som behöver komma åt sina datorer från Windows-maskiner.

Detta kommer att kopiera filen till den katalog där jag befinner mig då jag skriver kommandot, d.v.s. ".". Vill jag istället lägga filen jag hämtar i */tmp* skriver jag:

```
# scp fnord@foo.bar.org:foo.txt /tmp
```

scp använder sig i sin tur av **ssh** för att ansluta till den andra datorn, och är alltså inte i sig något självständigt program. Det behövs ingen speciell service på andra sidan anslutningen utöver den mottagande datorns **sshd**-demon (se nedan).

sftp är ett kommando som används för att skicka fler filer över en SSH-förbindelse. Det har en kommandosyntax som är densamma som för **ftp** och är enklare att använda för mer komplicerade filoperationer. Det finns också ett antal grafiska verktyg, t.ex. **gftp** som kan använda sig av **sftp** för att överföra filer.[49]

I likhet med **scp** är **sftp** ingen självständig tjänst: när du ansluter med programmet kommer **ssh** att användas för att ansluta till den andra datorn, men den mottagande datorns **sshd**-demon kommer att starta ett separat program med namnet **sftp-server** för att hantera kommandon som skickas av **sftp**. Ibland finns inte denna serverdel installerad, och då fungerar inte **sftp**.

sshd är den demon som körs på den mottagande datorn för att möjliggöra fjärrinloggning. Utan denna demon är det inte möjligt att använda SSH för att logga in på datorn, och eftersom de flesta vill kunna logga in på sina datorer utifrån brukar den vara installerad och startad. Vissa distributioner frågar om detta under installationen.

sshd kan konfigureras i filen */etc/sshd_config*, och det är i princip bara *en* viktig sak som ibland behöver ändras: raden *X11Forwarding* behöver ibland manuellt sättas till *yes*, vilket är nödvändigt för att kunna köra *X*-program över SSH och därmed via en krypterad förbindelse. Du kan också lägga märke till att **sshd** vid installation genererar ett antal nycklar för datorn, som också lagras i */etc/ssh* och som refereras från denna fil.

SSH bygger i grunden på den teori om digitala nycklar, signaturer och certifikat som är centrala inom datorsäkerheten och som inte kan avhandlas utförligt här. Vi skall dock lära oss lite om det praktiska handhavandet beträffande nycklar.

[49] För Microsoft Windows finns en populär SCP/SFTP-klient med namnet WinSCP, se `http://winscp.sourceforge.net/`.

Som tidigare sades kommer **sshd** under installationen att generera ett antal nycklar för att identifiera sig mot andra datorer. Nycklarna skickas över till de datorer som ansluter sig via nätverket, och första gången en anslutning sker kommer den också att sparas på den dator som ansluter sig, efter en fråga om huruvida nyckeln godkänns. Poängen med detta är att nyckeln sedan automatiskt kommer att jämföras vid senare anslutningar till samma dator, och skulle nyckeln skilja sig från den som tidigare erhållits kommer SSH att varna.

Det är dessvärre inte ovanligt att nycklar ändras av helt legala orsaker: den vanligaste är att datorn i andra änden har installerats om, och då har nya nycklar genererats, efterom systemadministratören på det systemet inte tänkte på att spara undan de gamla nycklarna i */etc/ssh* innan hon eller han installerade om systemet. Så gott som alla administratörer gör sig skyldiga till detta misstag.

När du som användare ansluter dig till en sådan ominstallerad dator kommer **ssh** att basunera ut varningar i stil med *it is possible that someone is doing something nasty* (vilket i och för sig är möjligt) och vägrar sedan att ansluta sig. Du får då ibland manuellt ta bort den gamla nyckeln, som lagras i filen *known_hosts* i den dolda katalogen *.ssh* i din hemkatalog. Gör detta genom att redigera *known_hosts* med en texteditor och ta bort raden för den aktuella datorn.

I den dolda *.ssh*-katalogen kan du också skapa en fil som heter *authorized_keys*. Denna används när du vill kunna logga in på en annan dator utan att behöva ange lösenord. Om du vill logga in från dator *A* till dator *B* skall du först köra programmet **ssh-keygen** på dator *A*:

```
# ssh-keygen -b 1024 -t rsa
```

Detta kommer att generera en RSA-nyckel[50] på 1024 bitar.

När programmet ber om ett lösenord brukar de flesta bara slå **ENTER**för att undvika att ange något lösenord — poängen med att använda nycklar för att identifiera sig är ju att slippa ange lösenord. Den genererade nyckeln med ett namnet *id_rsa.pub* skall sedan kopieras till dator *B*, där du skarvar till den i filen *~/.ssh/authorized_keys* med:

```
# cat rsa_id.pub >> ~/.ssh/authorized_keys
```

Du kan därefter ta bort den överförda filen, och logga ut. Vid nästa

[50]RSA är initialerna i efternamnen på uppfinnarna bakom algoritmen: Ron Rivest, Adi Shamir och Len Adleman och bygger på svårigheten med att faktorisera stora primtal. En normal rekommendation för att upprätthålla säkerheten i en RSA-nyckel är att den skall vara minst 1024 bitar lång.

inloggning från dator *A* kommer **sshd** på dator *B* att upptäcka att den datorn är godkänd, och släpper in dig utan lösenord.[51]

Men.

Detta öppnar ett stort, otäckt säkerhetshål i systemet och medför att den som obehörigen tar sig in i dator *A* genast kan ansluta sig till dator *B* precis som du själv kan göra detta utan att ange lösenord. Administratören av dator *B* kanske inte uppskattar detta och kan då helt stänga av funktionen att använda godkända nycklar i */etc/ssh/sshd_config*. Anledningen till att detta ändå är vanligt och oftast tillåtet är att de flesta är mindre oroliga för att obehöriga skall ta sig in i deras system: anledningen till att de använder SSH är att detta krypterar den överförda informationen, så att förbindelsen inte kan avlyssnas.

SSH kan också användas för s.k. *tunnling*. Detta går till så att en TCP-port på den egna datorn ansluts till en TCP-port på en annan dator, via en krypterad SSH-förbindelse. Via denna krypterade kanal kan sedan all form av ordinarie nätverkstrafik "tunnlas" mellan datorerna. Datorerna uppfattar detta som om de skickade data till sig själva. Tunnling får nog anses vara ett ganska avancerat sätt att använda SSH och sparas därför till ett annat tillfälle.

9.6 NTP

Officiell hemsida: `http://www.ntp.org/`

NTP, *Network Time Protocol*, är definierat av David Mills i RFC 1305 samt RFC 2030 och är avsett att synkronisera klockan i en dator med klockan i en annan dator, oftast via UDP-protokollet. Idén med detta är naturligtvis enkel: en dator frågar en annan dator — som helst skall vara ansluten till ett atomur — vad klockan är. När den andra datorn svarar, ställer den frågande datorn sin klocka.

Kruxet med detta är att hög precision fordras. Hög precision fordras speciellt i nätverk där användare delar filer: de flesta operativsystem stämplar sina filer med tid och datum, och skulle någon annan dator ha en annan klocka som går före, kan filer plötsligt vara daterade i framtiden, vilket kan inverka förvirrande på många datorprogram.

Detta är inte enda anledningen till att klockan i datorn helst skall gå rätt: många användare använder klockan i sin dator till allt möjligt,

[51] I detalj: dator *B* skickar ett meddelande krypterat med den publika nyckeln i *authorized_keys* och begär att dator *A* skall dekryptera detta, vilket den gör med hjälp av den privata nyckeln i *~/.ssh/id_rsa* och returnerar resultatet. Om resultatet matchar det ursprungliga meddelandet släpps du in. Själva meddelandet, som kallas för en *utmaning* (engelska: *challenge*) består bara av slumptal genererade av **sshd**.

och vill helt enkelt att den skall gå så rätt som möjligt. För människor krävs det inte lika hög precision som för program, men även i detta fall används naturligtvis NTP.

Hög precision är komplicerat att uppnå via IP-nätverk: det tar nämligen olika lång tid för paket att förflytta sig från en dator till en annan. För att kompensera för detta använder sig NTP av vissa trick som gör det möjligt att ställa klockan så att den slår på högst 10 millisekunder eller något ditåt. Hur detta går till kan vi lämna åt sidan i denna framställning.

Det rekommenderas att du använder NTP för att ställa klockan på din dator. Vissa distributioner erbjuder dig att automatiskt konfigurera systemet att använda NTP. Synkroniseringen skall utföras dels då systemet startas och dels med jämna intervall under körning, så att klockan inte ”drar sig”.

Sveriges Provnings- och Forskningsinstitut tillhandahåller mycket noggranna NTP-servrar installerade vid riksmätplatsen för tid och frekvens, och kan användas för att synkronisera datorer i Sverige. Dessa är: **ntp1.sp.se** och **ntp2.sp.se**.[52] Det finns även ett antal allmänt tillgängliga NTP-servrar på flera håll i landet. Ju kortare nätverksavstånd det är mellan dig och NTP-servern, desto bättre blir synkroniseringen. Följande serverar tillhandahålles av företaget Netnod och kan användas:

Göteborg	ntp1.gbg.netnod.se och ntp2.gbg.netnod.se
Malmö	ntp1.mmo.netnod.se och ntp2.mmo.netnod.se
Stockholm	ntp1.sth.netnod.se och ntp2.sth.netnod.se

Din nätverksleverantör kan ha stängt av tillgången till NTP-serverar med hjälp av en brandvägg,[53] och vill då oftast att du skall använda en lokal NTP-server istället. Kontakta i så fall din närmaste nätverksadministratör och fråga efter NTP-servern.

NTP-programmen för olika POSIX-system utvecklas direkt av samma personer som utvecklar protokollet, och programmen är samlade i ett paket som oftast bara heter *ntp*, ibland med programmet **ntpdate** utbrutet i ett eget paket med namnet *ntpdate*. Finns det senare är det antagligen det du som vanlig användare har mest nytta av, eftersom programmet **ntpdate** kan köras från kommandoraden för att ställa klockan på den egna datorn.

ntpdate körs tillsammans med ett servernamn, t.ex. **ntpdate ntp1.sp.se** för att ställa klockan i din dator mot den angivna NTP-servern.

[52]Se: `http://www.sp.se/metrology/timefreq/sv/tidssynk_ntp.htm`
[53]Se avsnitt 10.6 på sidan 360.

Programmet måste köras som root och ligger därför ibland i katalogen */usr/sbin* och kan eventuellt saknas i dina ordinarie sökvägar.

Programmet skall köras varje gång datorn startas. De flesta distributioner erbjuder en fil med namnet */etc/rc.local* där detta lämpligen görs, om NTP inte installeras automatiskt under installationen. Därutöver kan du sätta upp ett cronjobb[54] som körs som root och som synkroniserar mot tidsservern t.ex. en gång i timmen.

Ett bättre alternativ till att köra **ntpdate** titt som tätt är att instället starta demonen **ntpd**. NTP-demonen synkroniserar kontinuerligt datorns klocka mot en server enligt diverse regler. Bland annat detekterar demonen hur snabbt din dators klocka hamnar ur fas och synkroniserar med ledning av detta klockan mot en server lagom ofta.

Administrerar du någon form av lokalt nätverk bör du överväga att sätt upp en lokal NTP-server för ditt nätverk, som alla andra datorer i nätet i sin tur använder. På det viset uppnås hög lokal synkronisering mellan datorernas klockor. Din NTP-server bör sedan i sin tur synkroniseras mot en publik NTP-server på Internet.

Konfigurationsprotokollet DHCP kan automatiskt ange att en viss NTP-server skall användas för nätverket, detta används dock tyvärr mycket sällan.

[54]Se avsnitt 2.3.12 på sidan 107.

KAPITEL 10

Säkerhet

> Bara för att du är paranoid, behöver inte det betyda att de inte är ute efter dig.[1]
>
> — Kurt Cobain

Datasäkerhet (även kallat *datorsäkerhet*) är i största allmänhet ett omtalat ämne. Intresset för datasäkerhet tilltar kvadratiskt med storleken på den organisation vars integritet skall skyddas. Med *säkerhet* menas i detta sammanhang:

- Skydd av data som finns lagrad på datorn, t.ex. säkerhetskopiering av data så att en hårddiskkrasch inte förstör utfört arbete.
- Skydd mot driftstörningar, t.ex. mot fel som kan uppstå i programmen och förstöra utfört arbete eller orsaka kostsamma driftstopp.
- Skydd mot datorintrång, spioneri o.s.v. med mekaniska, tekniska och psykologiska medel.

Säkerhet har olika nivåer. Det säkraste datorsystemet är ett som inte alls kan användas. Därefter ett som är startat och inlåst i ett plåtrum,

[1]Min översättning av *Just because you're paranoid, don't mean they're not after you.*

utan nätverksanslutning, och med en eller flera heltidsanställda operatörer som tar hand om systemet och utför användarnas uppdrag. Eftersom stora datorsystem en gång verkligen fungerade på det viset, var de faktiskt mycket säkra.

Vad som menas med en ”lagom säkerhetsnivå” varierar. Föreläsaren i datasäkerhet på Lunds Tekniska Högskola, Ben Smeets, brukar ange att en sådan definition beror helt på hur paranoid du är. Paranoia är en schizofrenibesläktad psykisk sjukdom som ofta har som symptombild att den som drabbas tror sig vara förföljd av diverse ”fiender” eller ”hemliga organisationer”.

Säkerhetskraven tenderar att öka med storleken på det system som skall säkras. Datasäkerhet i en stor organisation är en betydligt allvarligare historia än den begränsade säkerhet som skall avhandlas här.[2] Syftet med detta avsnitt är att ge en enskild GNU/Linux-användare tillräcklig medvetenhet för att upprätthålla ett visst mått av säkerhet på den egna datorn.

10.1 Fysisk säkerhet

Att andra än betrodda användare kan använda en fysisk dator som de har tillgång till är ett direkt säkerhetshål. Du kan inte lita på någon av operativsystemets behörighetsfunktioner om någon användare har tillgång till systemkonsollen. (Den faktiska datorn, bildskärmen och tangentbordet.)

På sidan 186 talas det om hur lätt en användare kan starta datorn i enanvändarläge och därmed bli root på ett GNU/Linux-system. Detta är ett av de största säkerhetshål som existerar i Linux: under förutsättning att den som vill ta sig in i en dator har tillgång till maskinen, kan denne lätt ge parametern **single** till kärnan och är därmed root i systemet, eftersom detta förfarande normalt gör att systemet inte frågar efter något lösenord.

Detta säkerhetshål kan till viss del undvikas genom att ändra i filen **/etc/inittab** och under raden **id:N:initdefault:** (där N är standardkörnivån) lägga till raden **~~:S:wait:/sbin/sulogin**. I exempelvis Debian är detta gjort redan från början.

Detta är dock ingen fullständig medicin eftersom parametern **init=/bin/sh** har samma effekt som **single**. Istället bör du konfigurera ditt LILO eller GRUB så att det frågar efter ett lösenord om en användare vill ändra något. Detta hjälper, men bara under förutsättning att användaren inte kan stoppa in egna disketter eller CD-ROM i datorn. Skall även

[2] Se vidare sidan 417 om stordrift av GNU/Linux-system.

detta förhindras måste du stänga av möjligheten att starta från diskett eller CD-ROM i BIOS, och sedan har du inte mindre än tre olika lösenord till en och samma dator: BIOS-, GRUB- och root-lösenord.

Tänk på att svetsa igen datorlådan också, eftersom det går att nollställa BIOS, och därmed lösenordet, genom att pilla på moderkortet. Ungefär i detta läge är det hög tid att du frågar dig om du inte är lite väl paranoid. Det är stor risk att du faktiskt glömmer ett lösenord och lyckas låsa dig själv ute. Detta kan också läsas som: om du har en GNU/Linux-dator du av någon anledning inte kommer in i: nollställ BIOS och starta från diskett om inget annat hjälper.

Ett annat fysiskt skydd för den riktigt paranoide är att skärma av röjande signaler från datorn. Till exempel kan en bildskärm ”avlyssnas” på långt avstånd med speciell utrustning, och ”buggar” monteras i tangentbord och inuti datorer. Att placera hela datorsystemet i en Faradaybur löser problemet med röjande signaler. Den här typen av åtgärder är dock av militär säkerhetsgrad, och har du verkligen hot av militär grad eller tror dig vara i riskzonen för industrispionage kan du kanske börja med att säkerhetskontrollera alla människor som vistas i lokalen, innan du tillgriper denna typ av åtgärder.

Ibland skojas det ganska friskt med de som ägnar sig åt överdriven fysisk säkerhet, de kallas *aluminiumhattar* efter en grupp paranoiker i USA som enligt en vandringssägen isolerar sina hattar med aluminiumfolie för att undvika att få sina tankar avlyssnade av främmande agenter.

10.2 Uppdatering av systemet

För att hålla ett operativsystem säkert måste det med jämna mellanrum lappas ihop för att rätta till fel som upptäckts av tillverkaren.

Det engelska ordet *patch* betyder ungefär *lapp* eller *plåster*. När vi på svengelska talar om *patchning* av ett operativsystem är de *patchar*, eller *programlappar* det är fråga om avsedda att täppa igen en brist i systemet. Detta fungerar enligt samma princip som när en vulkaniserad gummilapp används för att laga ett cykeldäck, eller så som ”armbågslappar” av läder på tröjor gör det möjligt att använda dem trots att det blivit hål på armbågarna. Ett annat namn på dessa programlappar är kort och gott *programuppdateringar* eller *systemuppdateringar*.

De flesta GNU/Linux-distributioner släpper med jämna mellanrum sådana lappar, för att rätta till brister i programvaran som upptäckts efter att installationsmedier redan distribuerats. Förr i världen kom dessa på kassettband, sedan på CD-ROM-skivor, och uppdateringsfrekvensen kunde vara någon gång var sjätte månad eller liknande.

Nuförtiden är sådana programlappar mycket mera frekventa: de distribueras via Internet och installeras ibland rent av automatiskt. De gamla systemen med manuell uppdatering från CD-ROM-skivor används i princip bara för uppgradering av en hel distribution från en version till en annan, och knappt ens då.

Emedan en del programlappar helt enkelt åtgärdar enkla användarfel, som gör det svårt att använda systemet eller orsakar obehag — t.ex. om ett populärt ordbehandlingsprogram tenderar att krascha mitt i en viss operation — är andra avsedda att rätta till allvarliga säkerhetsbrister i systemet, som kan göra att andra kan ta över kontrollen av din dator via nätverket.

Du bör, för att hålla ditt system säkert i detta avseende:

- Veta hur ditt GNU/Linux-system skall uppdateras på enklaste sätt.
- Ha någon form av informationskanal som snabbt upplyser dig om när det är dags att installera programlappar, i synnerhet om dessa på något vis är säkerhetsrelaterade, t.ex. via epost, en webbsida du dagligen besöker, eller ett datorprogram som körs automatiskt och upplyser om förändringen. Det senare är att föredra.

Flera distributioner innehåller system för att hålla datorn uppdaterad med nya programlappar. I Debian GNU/Linux finns APT-systemet och i Red Hat Enterprise Linux finns Up2date-funktionen. Dessa hjälper till att se till att de senaste uppdateringarna alltid är installerade, men kräver för det mesta att användaren aktivt väljer att utföra åtgärden, så att en manuell rutin krävs för varje ny uppdatering.

Vissa tar sig för att lägga in cronjobb[3] som automatiskt uppdaterar operativsystemet med alla programlappar som kommer från distributionen. Exempelvis kan crondemonen köra ett skript på en Red Hat-installation som en gång om dygnet kör **up2date -u** eller på en Debian-installation kör **apt-get update -q -y ; apt-get upgrade -q -y**. Detta kommer kontinuerligt att hålla maskinen igenlappad, men är lite smått farligt, i vart fall gällande Debian "testing" som lite då och då får in trasiga saker i sin paketstruktur, något som kan ställa till det rejält.

10.3 Säkerhetskopiering

Säkerhetskopiering (engelska: *backup*) innebär att du gör kopior av alla viktiga filer på en dators hårddisk. Det kan även röra sig om att göra

[3] Se sidan 107.

kopior av alla filer på en centralt belägen filserver. Dessa kopior görs för att data ibland går förlorade, dels för att hårddiskar går sönder, och dels på grund av användarfel, d.v.s. att du tar bort något av misstag eller med flit, som det sedan visar sig att du behöver. Datorvirus eller datorintrång kan också leda till att data förstörs, liksom inbrott och stöld.

Grunden till all säkerhetskopiering är inte teknisk utan en rutin som beskriver vad som skall kopieras, hur ofta det skall göras, och vart kopiorna skall sparas. Misslyckandet med att följa dessa ganska självklara rutiner är lustigt nog den vanligaste anledningen till att säkerhetskopior inte finns när de behövs.

I POSIX-system görs säkerhetskopior traditionellt sett med kommandona **tar**,[4] **compress** och **gzip**. (Se sidan 59 för beskrivning.) Själva namnet *tape archive* skvallrar en del om vad kommandot är till för. Förr gjordes en säkerhetskopia helt enkelt i form av en stor tar-fil av det som var viktigt att spara, som sedan kopierades till ett band.

I filsystemet Ext2 (och Ext3) finns ett inbyggt kommando kallat **dump** som kan göra en binär kopia av ett helt filsystem. Detta kan vara bra för mindre filsystem, men inte för stora system: att spara allt ur ett filsystem tar mycket plats, men är bra om du vill göra en fullständig kopia av ett systems hårddisk, och dessutom har någonstans att lägga den. **dump -0 /dev/hda1 -f foo.dump** kommer t.ex. att göra en säkerhetskopia av första partitionen på IDE-hårddisk 1 till en fil med namnet *foo.dump*.[5] Det går även att göra en säkerhetskopia till exempel till ett flertal disketter med **dump -0 /dev/hda1 -f /dev/fd0**.

En fördel med **dump** är att det inte påverkar filsystemet på något vis: **tar** kommer om inte annat att påverka tiddstämpeln som anger den tid då en viss fil sist lästes. För att återställa ett filsystem används **restore** som är omvändningen till **dump**. Om du nu inte använder Ext2/3 är detta inte något alternativ för dig. Kommandot kräver dessutom att filsystemet inte är monterat i skrivbart läge när det dumpas.[6]

Ett möjligt sätt att tillgodose behovet av säkerhetskopiering är att hålla sig med dubletter. Du kan t.ex. köpa en extra hårddisk som är exakt lika stor som din vanliga, och regelbundet starta datorn i enanvändarläge och kopiera allt till denna disk med kommandot **dd if=/dev/hda of=/dev/hdb**. (Var nu noggrann med att *if* är den disk du skall kopiera *från* och *of* den disk du skall kopiera *till*!)

Detta sätt att använda två hårddiskar är dock inte så smart. Det na-

[4]Det finns även en nyare variant av **tar** med namnet **star**, som även kan hantera bl.a. accesskontrollistor.

[5]Gör inte detta så att dumpfilen hamnar på samma partition som du gör en dump av, det blir rundgång då och hårddisken fylls snabbt.

[6]**mount -o remount,ro foo**

turliga sättet att använda en extra hårddisk är att köra RAID 1,[7] vilket innebär att all information som går genom systemet automatiskt skrivs till båda hårddiskarna. Detta ger dock bara skydd mot hårddiskkrascher, inte mot katastrofer som orsakas av att stora mängder filer raderats.

10.3.1 Vad?

Det är lätt att slänga ur sig att hela datorns innehåll skall säkerhetskopieras, men svårt att genomföra det, ofta p.g.a. utrymmesbrist. Följande delar av filsystemet skall i princip *alltid* säkerhetskopieras:

/home — här ligger ju all data du och andra användare av systemet har skapat! Slarva *aldrig* med att säkerhetskopiera */home*!

/etc — här ligger ju en hel del inställningar som du gjort i systemet efter installationen. Även om du inte tänker kopiera tillbaka */etc* om t.ex. hårddisken kraschar och du får installera om systemet, så kan det vara skönt att kunna kontrollera hur du gjorde inställningarna sist.

/usr/local — här ligger alla program du har installerat själv. Synd att bli av med dem om något skulle hända.

/opt — där programpaket från tredje part ibland har installerats med mycket möda och stort besvär.

/var — är platsen där databaser, epost och liknande brukar hamna. Här är det dock lite kluvet: distributionerna lägger t.ex. sina listor över installerade programpaket här, och har du då inte också kopierat */usr*-hierarkin och resten av rotfilsystemet, t.ex */bin* så kommer systemet att bli instabilt ifall bara */var* kopieras tillbaka till en ny installation. Det sparas dessutom en hel del loggar och liknande saker i */var* som inte alltid är så ovärderliga. Välj med omdöme.

En del väljer därför att bara kopiera */home* och konstatera att: ”om nåt händer, då får jag installera om systemet och alla mina program”. Denna strategi är helt acceptabel om operativsystemet installerats standardmässigt, t.ex. ett Red Hat Linux-system rakt av från hyllan, utan krusiduller.

I stora organisationer är */home* oftast monterad på en nätverksenhet med t.ex. NFS, medan klienten är en ren standardinstallation. I sådana

[7] Se vidare avsnittet B.3 om RAID på sidan 421.

fall kan en trasig eller problematisk dator helt enkelt installeras om, eftersom ingen användardata skall ligga på själva arbetsdatorn över huvud taget. NFS-servern däremot, som lagrar alla användarfiler, måste säkerhetskopieras desto noggrannare.

10.3.2 Hur ofta?

Rutiner för säkerhetskopiering har utkristalliserat sig under historiens gång.

När säkerhetskopieringsarbetet blev mer komplext var den naturliga åtgärden att göra skript som utförde arbetet och gjorde säkerhetskopior av olika saker och med olika intervall. Alla som skrev sådana skript hade egna idéer om hur mycket som var lagom att kopiera och hur ofta det skulle göras. En vanlig rutin som tillämpas är:

En gång i månaden skall allt viktigt i hela systemet säkerhetskopieras, rubb som stubb.

En gång i veckan görs en säkerhetskopia av det som ändrat sig sedan månadsskiftet.

En gång om dagen t.ex. på kvällen, görs en säkerhetskopia av det som ändrat sig sedan veckoskiftet.

De olika säkerhetskopiorna sparas sedan en längre tid, tills du är helt säker på att de inte behövs längre, eller för alltid. Är det ett medium som kan användas flera gånger som används för säkerhetskopieringen, så kan det naturligtvis raderas och återanvändas. (Omskrivbara CD-ROM- och DVD-skivor till exempel.)

Att göra säkerhetskopior som bara innehåller det som ändrats jämfört med någon tidigare kopia kallas *inkrementell kopiering* eller *flernivåkopiering* om fler än två nivåer är tillgängliga. Den här föreslagna metodiken har tre nivåer: månad, vecka och dag. Den högsta nivån symboliserar alltid en fullständig kopia av allt som ska säkerhetskopieras i systemet. Det tidigare nämnda kommandot **dump** för Ext2/3-system stödjer upp till nio olika nivåer, men jag har aldrig hört talas om någon som använder så många nivåer.

Om du bara gör säkerhetskopior för ditt eget vidkommande, t.ex. för hemmabruk, kanske detta känns lite överdrivet. Kanske upplever du det också som onödigt att packa samman filerna i ett *tar.gz*-paket, och nöjer dig med att helt enkelt bränna alla viktiga filer på en CD- eller DVD-skiva.

10.3.3 Hdup

Officiell hemsida:
`http://www.miek.nl/projects/hdup16/`

Om du nu inte nöjer dig med att använda **tar** och egna skript för att göra säkerhetskopior så finns det även verktyg att ta till. Det förmodligen mest populära är *Hdup*. Detta program finns dessvärre sällan med i vanliga distributioner, så du är tvungen att ladda ner och kompilera det själv, om du inte kan hitta ett färdigt paket.

Hdup är litet och simpelt. Det bygger på en konfigurationsfil som lagras i */etc/hdup/hdup.conf* där du ställer in hur och var säkerhetskopior skall göras. I denna fil finns först en global inställning som anger värden som gäller alla säkerhetskopior, och sedan en sektion för varje kopieprofil, som är en egen typ av säkerhetskopia, t.ex. på följande vis:

```
[foo]
dir = /etc /home
```

Du kör sedan en månatlig säkerhetskopia med kommandot **hdup monthly foo**. Eftersom programmet Hdup har en tendens att installera sig självt i */usr/local/sbin* kan du behöva ange hela sökvägen, eller lägga till denna plats i miljövariabeln **$PATH**.

När du konfigurerat och testat Hdup på din maskin kan du automatisera det genom att lägga in tre rader i din *crontab*-fil[8] så att programmet körs en gång i månaden, en gång i veckan och en gång om dagen:

```
0 0 1 * * /usr/local/sbin/hdup monthly foo
0 1 * * 1 /usr/local/sbin/hdup weekly foo
0 18 * * * /usr/local/sbin/hdup daily foo
```

Denna crontabell kommer att göra säkerhetskopior enligt profilen *foo* ovan. Notera att den dagliga kopian görs klockan 18, så att varje dags arbete finns sparat. Kopiorna hamnar på en viss plats i filsystemet, så om detta är exempelvis en bandstation, CD-R-brännare eller en extra hårddisk, kommer säkerhetskopiorna att underhålla sig själva där.

Om något sedan skulle inträffa, t.ex. en hårddiskkrasch, kan alla filer återskapas med **hdup restore foo 2003-06-06 /tmp** ifall den senaste kopian av *foo* gjordes den sjätte juni. När */tmp* anges på det här viset kommer kopiorna att dyka upp i */tmp* som (i detta fall) */tmp/home* och

[8] Se avsnittet 2.3.12 om cronjobb på sidan 107.

/tmp/etc. Du kan då först kontrollera att de är korrekta innan du kopierar tillbaka dem (eller delar av dem) till den plats varifrån de kopierades.

Hdup kan även göra säkerhetskopior över nätverk, vilket är det bästa sättet att använda programmet på om du har tillgång till mer än en dator. Detta fordrar dock en del nyckelutbyte med SSH[9] så att Hdup kan logga in utan att skriva in något lösenord. Detta är *i sig* ett säkerhetshål, men kanske ett du är villig att acceptera för en annan form av säkerhet.

Om du kör ett system där du använder accesskontrollistor är det viktigt att du gör säkerhetskopiorna med programmet **star** istället för vanliga **tar**, som inte klarar av att hantera accesskontrollistor. Vilket filarkiveringsprogram som Hdup skall använda går också att modifiera i */etc/hdup/hdup.conf*.

Förutom Hdup finns en mängd s.k. ”professionella” säkerhetskopieringssystem av varierande sofistikation. Ofta är de proprietära, ganska dyrbara, och i ett stort system krävs dessutom speciell hårdvara, t.ex. robotar som automatiskt kan byta mellan ett stort antal kassetter för att rymma stora mängder säkerhetskopior.

10.3.4 Rsync

Officiell hemsida: `http://rsync.samba.org/`

Rsync är ett program utvecklat av Andrew James Tridgell i syfte att tillhandhålla ett verktyg som kunde synkronisera innehållet i olika filsystem över ett datornätverk. Detta program används till flera ändamål, t.ex. för att publicera webbsajter från ett internt nätverk till ett öppet, och diverse synkronisering av arbetsdata över ”stora nätverksavstånd”. Med ”stort nätverksavstånd” menas inte geografiskt avstånd, utan att datorerna har ett långsamt nätverk mellan sig. Detta kan i och för sig bero på långt geografiskt avstånd, men kan också bero på annat.

I säkerhetssyfte är ett av huvudanvändningsområdena för Rsync att kopiera hela, eller delar av ett filsystem mellan två eller flera datorer för att på så vis ”spegla” innehållet i ett filsystem för att sprida riskerna. Ett typiskt sådant användningsområde är att synkronisera en bärbar dator med en stationär, så att filerna på den ena datorn också finns på den andra. Detta fungerar som en säkerhetskopia genom att data i filsystemet dubbleras. Blir vi av med den ena datorn finns den andra kvar. Vi skall nu titta närmare på fallet *hemdator* och *bärbar dator*, och visa hur vi kan synkronisera innehållet på dem.

[9] Se avsnitt 9.5 på sidan 331

Rsync används på olika sätt. Det enklaste är en envägssynkronisering, från en källa till en destination, vilket är lämpligt om du bara vill ha en kopia av alla saker från en viss katalog på din hemdator med dig på din bärbara dator. Rsync-kommandot har som så många andra POSIX-kommandon formen:

rsync [-växlar] KÄLLA MÅL

Källan är platsen det skall synkroniseras *från* och *målet* är platsen det skall synkroniseras *till*. Så om källan är en katalog på den datorn där du har öppnat ditt skal, och målet är en katalog på en dator med namnet *laptop.bar.org* skriver du till exempel:

```
rsync -avrz -e ssh /home/eva/mp3 eva@laptop.bar.org:/home/eva
```

Detta kommer första gången att göra en kopia av katalogen */home/eva/mp3* från den datorn där det körs, till datorn med namnet *laptop.bar.org*, i användarkontot *eva*, och på samma plats.[10] Växlarna **-avrz** talar om att Rsync skall köras i ”arkiveringsmod”, vilket betyer ungefär ”kopiera allt”, att vi vill ha mycket utskrifter från programet (*verbose*), att katalogen *foo* skall synkroniseras rekursivt, d.v.s. att även underkataloger och filer och kataloger som finns i dessa skall kopieras och synkroniseras, samt att datan som överförs skall komprimeras med GNU Zip under transporten. *-e ssh* ber programmet att använda SSH[11] för att överföra informationen, så att det hela krypteras.

Nästa gång du kör samma kommando kommer kopian på den andra maskinen att uppdateras, och då kommer Rsync bara att överföra den ändrade informationen — filer som inte har ändrats sedan sist kommer inte att överföras, och om du bara har ändrat lite grand mitt i en fil, kommer bara den ändrade biten att överföras och kopieras in i filen på den andra datorn. Detta fungerar även på binära filer som t.ex. ordbehandlingsdokument, så i allmänhet går synkroniseringar mycket snabbt, det är bara under den första synkroniseringen som det tar tid.

En förklaring rörande syntaxen för synkronisering av kataloger kan vara på sin plats: att ange t.ex. */home/eva/mp3* som källa eller destination i ett Rsync-kommando innebär att katalogen *mp3* (och alla kataloger inunder den om du arbetar med växeln **-r**) kommer att synkroniseras, om

[10] I exemplet används en fullständig sökväg till användarkontot, detta är inte nödvändigt: **rsync -avrz -e ssh mp3 eva@laptop.bar.org:** räcker utmärkt om du ursprungligen står i din hemkatalog, men detta skrivsätt brukar ge större känsla av trygghet och fungerar även när du inte vet vart du befinner dig i filsystemet. Om du inte vet platsen för ett användarkonto på en annan maskin kan du helt enkelt logga in på den och skriva **pwd**, i detta fall var det exakt samma plats som på originalmaskinen.

[11] Se avsnitt 9.5 på sidan 331

/home/eva/mp3 är en katalog (det skulle ju kunna vara en vanlig fil som heter *mp3*). Om du istället skriver */home/eva/mp3/* som källa, kommer alla filer och kataloger *under* katalogen *mp3* att synkroniseras istället, så om */home/eva/mp3* innehåller katalogerna *foo* och *bar* kommer kommandot:

```
rsync -avrz -e ssh /home/eva/mp3/ eva@laptop.bar.org:/home/eva
```

inte att skapa katalogen */home/eva/mp3* på datorn *laptop.bar.org* (vilket säkert var avsikten), utan istället katalogerna */home/eva/foo* och */home/eva/bar* med respektive innehåll, samt kopiera alla filer som låg direkt i */home/eva/mp3* direkt ner i katalogen */home/eva*! Det blir ganska jobbigt att rensa upp efter ett sådant misstag, även om normalt ingenting direkt skadas.

Eftersom Rsync-synkroniseringar går så pass snabbt går det bra att köra dem ofta, t.ex. var tionde minut om du har två vanliga hemdatorer, som du vill hålla i synk. De är också bra för modemuppringda förbindelser, eftersom mängden information som överförs är så pass låg att det inte kostar för mycket tid att hålla t.ex. sin dator hemma och en bärbar dator i synk.

Du vill kanske köra synkronisering i båda riktningarna, så att det du skapat på din bärbara dator finns kopierat på din hemdator och vice versa. Då kan du förstås bara köra exakt samma kommando som ovan från den bärbara datorn, men det går också att initiera en överföring i båda riktningarna från en och samma plats:

```
rsync -avrz -e ssh eva@laptop.bar.org:/home/eva/mp3 /home/eva
rsync -avrz -e ssh /home/eva/mp3 eva@laptop.bar.org:/home/eva
```

Detta kommer att synkronisera katalogen *foo* först i riktningen bärbar dator → hemdator, och därefter i riktningen hemdator → bärbar dator.

Om du tar bort filer från den ena datorn kommer Rsync standardmässigt inte att utsätta dig för några risker: inga filer raderas från en dator som synkroniseras med Rsync, förrän du anger växeln **--delete**. Således kan användaren *eva* i exemplet lägga några nya filer i katalogen */home/eva/mp3* på sin bärbara dator när hon är ute och reser, och de kommer då att kopieras till hennes hemdator när hon synkroniserar med ovanstående två kommandon. På samma vis kan hon lägga nya filer i */home/eva/mp3* på sin hemdator och de kommer då att kopieras till hennes bärbara dator. Detta är helt idealiskt när det bara är fråga om att lägga till eller ta bort filer.

Att använda --delete i riktningen hemdator → bärbar dator är bra om du bara vill ha en exakt kopia av en katalog med dig när du reser — filer kommer då att raderas så att katalogen på den bärbara datorn ser ut exakt som den på din hemdator.

Men detta är *inte bra* om du lägger till filer på din bärbara dator, för då kommer de omedelbart att raderas ifall du kör **rsync --delete** från hemdatorn!

Detsamma gäller *alltid* om du går in och ändrar i en fil på din bärbara dator, och först synkroniserar i riktningen hemdator → bärbar dator — då kommer eventuella ändringar du gjort på din bärbara dator att skrivas över av originalet från din hemdator! På exakt samma vis kan du skjuta dig i foten genom att synkronisera i riktningen bärbar dator → hemdator först (som i exemplet ovan): om du har ändrat i filer på din hemdator kommer ändringarna att försvinna! Så *se upp!*.

För att sköta detta på ett vis som inte utsätter dig för risker fordras som så ofta en *rutin*, d.v.s. ordning och reda. Jag brukar se till att:

- Synkronisera bara i ena riktningen, med **--delete** för kataloger som jag *bara vill ha med mig* men inte ändrar i.
- Synkronisera filer i båda riktningarna för kataloger där jag bara *lägger till* saker när jag är ute och reser.
- Skapa *en speciell katalog*, t.ex. */home/eva/laptop* som är till för saker som jag vill arbeta med på min bärbara dator. Denna specialkatalog fungerar så att den bärbara datorn är primär, d.v.s. ändringar från den bärbara datorn skriver över alla saker på hemdatorn, men om jag bara *lägger till* en fil på hemdatorn, kommer denna att kopieras över till den bärbara så att jag kan arbeta med den där sedan. Här lägger du saker som du ska *arbeta med* när du är på resande fot.

Jag har dessutom *två olika skript*: ett som jag kör på min hemdator när jag synkroniserar den bärbara datorn hemma (vilket nog räcker för många) och som återfinns i figur 10.1, och ett som jag kan köra "från fältet" så att jag säkerhetskopierar saker hem, och som återfinns i figur 10.2. (Datorn där hemma får naturligtvis alltid vara påslagen för att det senare skall fungera.)

Exempelskripten i figur 10.1 och 10.2 arbetar relativt den angivna användarens hemkatalog på destinationsdatorn, och kommer i exemplet att synkronisera i ena riktningen (*till* datorn som angetts i LAPTOP-variabeln) de kataloger som angetts i variabeln COPYDIRS, i *båda* riktningarna med de kataloger som angetts i SYNCDIRS och med prioritet för den bärbara datorn med den katalog som angetts i WORKDIR.

```
#!/bin/sh
LAPTOP="laptop.bar.org"
LAPUSER="sofie"
COPYDIRS="bin foo"
SYNCDIRS="mp3"
WORKDIR="laptop"

# Kopiera i ena riktningen
for DIR in ${COPYDIRS}; do
    rsync -avrz --delete -e ssh ${HOME}/${DIR} \
        ${LAPUSER}@${LAPTOP}:
done;

# Synkroniserade saker
for DIR in ${SYNCDIRS}; do
    rsync -avrz -e ssh ${HOME}/${DIR} ${LAPUSER}@${LAPTOP}:
    rsync -avrz -e ssh ${LAPUSER}@${LAPTOP}:${DIR} ${HOME}
done;

# Arbetskatalog för bärbar dator
rsync -avrz -e ssh ${LAPUSER}@${LAPTOP}:${WORKDIR} ${HOME}
rsync -avrz -e ssh ${HOME}/${WORKDIR} ${LAPUSER}@${LAPTOP}:
```

Figur 10.1: Skript som synkroniserar i riktningen hemdator → bärbar dator.

```
#!/bin/sh
DESKTOP="desktop.bar.org"
DESKUSER="sofie"
SYNCDIRS="mp3"
WORKDIR="laptop"

# Synkroniserade saker
for DIR in ${SYNCDIRS}; do
    rsync -avrz -e ssh ${HOME}/${DIR} ${DESKUSER}@${DESKTOP}:
    rsync -avrz -e ssh ${DESKUSER}@${DESKTOP}:${DIR} ${HOME}
done;

# Arbetskatalog för bärbar dator
rsync -avrz -e ssh ${HOME}/${WORKDIR} ${DESKUSER}@${DESKTOP}:
rsync -avrz -e ssh ${DESKUSER}@${DESKTOP}:${WORKDIR} ${HOME}
```

Figur 10.2: Skript som är avsett att användas på resande fot, och som synkroniserar i riktningen bärbar dator → hemdator.

Rsync kan användas för mycket utöver att synkronisera en bärbar dator med en hemdator, men detta exempel ger en god uppfattning om de olika saker du kan behöva göra med Rsync.

I praktiken kommer exemplen här, liksom skripten att kräva att du anger ett lösenord till den andra datorn vid varje körning av kommandot **rsync**. Du vill nog i normalfallet slippa detta, genom att använda godkända inloggningsnycklar i SSH, så att du inte behöver skriva ett lösenord varje gång du synkroniserar en katalog. Om du skall köra Rsync mellan t.ex. två hemdatorer, vill du nog köra det med hjälp av ett cronjobb, se sidan 107.

En lustighet med Rsync, liksom med **rcp** och **scp**, är att det inte får förekomma några utskrifter av text i skalet på den dator du kör kommandot mot. Om du t.ex. lagt in raden **echo "Fnord"** i din *~/.bashrc*-fil, kommer det att ge upphov till diverse klagomål och fel och Rsync kommer inte att fungera alls, så låt för all del bli att i ditt standardskal ha program eller funktioner som skriver ut saker i skalet varje gång ett nytt skal startas, det vållar nämligen bara problem.

Ett vanligt missförstånd rörande Rsync är att tro att detta verktyg kan synkronisera i båda riktningarna, och så att säga slå samman ändringarna i två filer. Det kan det *inte*. För en människa är detta en trivial sak, men för en dator på gränsen till omöjligt, eftersom det kräver viss intelligens. För rena textfiler med programkällkod finns system[12] som löser vissa delar av detta behov, men inte utan att göra fel ibland. När det gäller saker som bilder och ordehandlingsdokument är en sådan sak än mer komplex.[13] Rsync bygger helt på att en källa skriver över ett mål, inte på sammanslagning av de två. Har en fil ändrats på båda ställena, kommer den som synkroniseras först att skriva över den andra, vars ändringar således försvinner.

10.4 Kryptering

Kryptering är processen att förvränga information. I de sammanhang som presenteras här är kryperingen vanligtvis av enklaste slag: ett block information av fast storlek förvrängs med en viss *algoritm*, kopplad till en viss *nyckel*, så att bara den som har tillgång till nyckeln kan dekryptera och läsa informationsblocket.

[12] Versionshanteringssystem såsom CVS eller Subversion.

[13] För strukturerad information som redovisningskonton och stora dynamiska tidtabeller används databaser för att åstadkomma denna typ av funktionalitet, när information behöver uppdateras från flera ställen samtidigt. Då används inte enskilda filer med information över huvud taget.

Nyckeln kan vara en fil på en hårddisk, diskett o.s.v. Oftast är denna nyckel ganska stor. För att skydda nyckeln krypteras denna oftast i sin tur med ett lösenord. Ibland används ett lösenord direkt för att generera en större nyckel, något som brukar kallas *hashing*.[14]

Kryptering på ett GNU/Linux-system kan skötas av såväl enskilda applikationer som av Linuxkärnan, förutsatt att kärnan har kompilerats så att den har stöd för kryptering. Krypteringen i kärnan var länge en het potatis, eftersom teknologi av detta slag betraktades som militär utrustning, och således kom att omfattas av exportrestriktioner. Sedemera har dessa restriktioner i flera länder successivt avskaffats, i synnerhet i USA, som länge var tveksamt till spridning av krypteringsteknologi.

I och med version 2.6 av Linuxkärnan inkluderades därför slutligen kryptering i kärnan. I skrivande stund har detta stöd ännu inte stabiliserat sig, och det är därför oklart om och hur omkringliggande verktyg skall använda krypteringsstödet i kärnan.

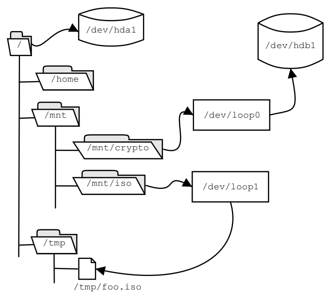
Eftersom *import* av kryptoteknik inte är något problem i Sverige, kan vi med gott samvete använda vilken krypteringsalgoritm vi vill. Om du däremot t.ex. reser med en bärbar dator in och ut ur Sverige uppstår en mängd juridiska problem, även om du bara använder kryptot för att skydda din egen dator. De flesta sansade människor bryr sig inte om detta, men som ansvarig för en större organisation bör du ta dig en funderare och kontakta en jurist om du planerar stora installationer av gränsöverskridande system.

10.4.1 Kryptering av hårddisk

Om du inte lyckas skydda din dator fysiskt, exempelvis om du har en bärbar dator, kan det vara värt att kryptera hårddisken. Detta hindrar inte andra från att radera din hårddisk och använda datorn till något annat om den t.ex. skulle bli stulen (du har väl säkerhetskopior??), men det hindrar dem effektivt från att plocka ut någon information från hårddisken.

Kryptering av hela hårddiskar i Linux brukar åstadkommas med det snillrika *loopback*-gränssnittet. Detta gränssnitt gör det möjligt för saker som inte är blockenheter att bete sig *som om de vore* blockenheter. Det vanligaste användningsområdet för dessa ”loopar” är att skapa en ”låtsashårddisk” eller ”låtsas-CD-ROM” genom att montera en fil som om den vore en blockenhet. En hel partition kan också monteras som

[14] Hashfunktioner brukar normalt mappa ett element ur en större finit domän på ett element i en mindre finit domän, men i detta fall är användningen den omvända. Ordet *hash* betyder *mala sönder* eller *söndermalet*. *Pölsa* (d.v.s. malda inälvor) heter också *hash* på Engelska.



Figur 10.3: Exempel på *loopback*-montering. I figuren är en krypterande loop ansluten mellan monteringspunkten */mnt/crypto* och partitionen */dev/hdb1*, och en mellan monteringspunkten */mnt/iso* och filen */tmp/foo.iso*.

en loop, vad nu nyttan skulle vara med det. För en illustration, se figur 10.3.

Mellan en loop och en lagringsplats, oavsett om det är fråga om en fil eller en partition, kan ett krypterande lager införas, så att all information som skrivs eller läses från lagringsplatsen kommer att krypteras på vägen in och ut. På detta vis kan antingen en hel partition eller en fil som får motsvara en lagringsenhet krypteras.

Den modul i version 2.6 av Linuxkärnan som har hand om krypterade loopar heter *cryptoloop* och måste vara inkompilerad i din kärna, eller åtminstone tillgänglig som modul. Om modulen inte skulle vara laddad får du naturligtvis antingen editera dina uppstartskript så att detta görs automatiskt, eller skriva **modprobe cryptoloop** för att göra det för hand. Andra moduler som måste finnas är den vanliga *loopback*-modulen och naturligtvis kryptomodulerna (t.ex **modprobe aes** för AES-kryptot).

Ett ytterligare problem med 2.6-kärnan är att paketet *Util-linux* där bland annat **mount** finns, länge saknade stöd för krypterade loopbackenheter. Om din version av *Util-linux* är version 2.12 eller äldre har du sannolikt detta problem, och får leta reda på patchar[15] för att få det hela att fungera. I senare versioner är problemet avhjälpt, och du som läser detta använder kanske redan en modern distribution där allt detta fungerar som smort.

I version 2.4 av kärnan finns sällan stöd för krypterade loopar i de distributioner jag sett, men många som är i behov av kryptera hård-

[15] I Cryptoloop-HOWTO finns länkar till lämpliga patchar och uppdaterad information, se: `http://www.tldp.org/HOWTO/Cryptoloop-HOWTO/`

disk väljer att använda *loop-AES*,[16] vilket är en separat modul till kärnan, som kräver modifikation av kärnans källkod och av de till kärnan hörande verktygen, t.ex. **mount**. Kryptering av hårddiskar under 2.4-kärnan är således ingenting som kan göras lättvindigt. I detta avsnitt kommer det inbyggda stödet i 2.6-kärnan att presenteras, men principen är densamma för loop-AES.

Monteringspunkt
⇕
/dev/loop0
⇕
Krypteringslager
⇕
Lagringsutrymme: partition eller fil

Att endast ha en enstaka fil med krypterad data är användbart om du bara har ett fåtal filer som du vill hålla hemliga, men att kryptera åtminstone hela den partition där */home* ligger (vilket ofta är rotpartitionen, */*) rekommenderas om du vill skydda din användardata ordentligt. Eftersom bara root får montera enheter bör du vara root på det system du skall skydda med hårddiskkryptering.

För att skapa en 10 megabyte stor fil med namnet */var/foo.img*, innehållande ett krypterat Ext2-filsystem, som använder kryptot AES,[17] monterad med loopback som katalogen */mnt/krypto* kan följande kommandosekvens användas:

```
# dd if=/dev/urandom of=/var/foo.img bs=1 count=10M
# losetup -e aes /dev/loop0 /var/foo.img
# mkfs -t ext2 /dev/loop0
# mkdir /mnt/krypto
# mount -t ext2 /dev/loop0 /mnt/krypto
```

Detta skapar först en 10 megabyte stor fil med slumptal. Slumptalen är viktiga, eftersom en angripare som vill dekryptera filen annars kan se vilka block som används av filsystemet. Sedan monteras filen med loopback-gränssnittet */dev/loop0*, och samtidigt anger vi att allt som loopas genom detta gränssnitt skall krypteras med AES-kryptot. **losetup** kommer att fråga efter ett lösenord: detta skall vara så långt och svårgissat som möjligt.

[16]Se: `http://loop-aes.sourceforge.net/`

[17]AES utläses *Advanced Encryption Standard* och är efterföljaren till den tidigare DES *Digital Encryption Standard*. AES hette tidigare Rijndael och valdes som efterföljare till DES av en amerikansk standardkommitté.

Efter detta skapas ett Ext2-filsystem på blockenheten */dev/loop0* som sedan monteras precis som vanligt. Du kan lika gärna ange en blockenhetsfil till **losetup** — t.ex. */dev/hdb* för att göra hela den andra hårddisken till en krypterad enhet.

För att avmontera den krypterade filen eller enheten efter att den använts för första gången skriver du:

```
# umount /mnt/krypto
# losetup -d /dev/loop0
```

Vill du senare montera den krypterade enheten igen, kan du skriva:

```
mount -o loop,encryption=aes /var/foo.img /mnt/krypto/
```

Och avmontera med **umount** på vanligt vis. Systemet kommer automatiskt att starta och avsluta loopback-gränssnitt vid behov. Det går också utmärkt att lägga in en rad i */etc/fstab* på denna form:

```
/var/foo.img  /mnt/krypto  ext2  noauto,encryption=aes  0 0
```

Efter detta går det utmärkt att montera det krypterade filsystemet bara genom att skriva **mount /mnt/krypto**, och avmontera det med **umount /mnt/krypto**. Vid montering kommer du att frågas efter lösenordet, och detta kommer sedan att användas för att låsa upp den krypterade filen. Uppsättningen och borttagningen av ett loopback-gränssnitt för kryperingen sker automatiskt. Nyckelordet *noauto* talar om för systemet att det inte skall försöka montera denna enhet under uppstart; istället måste du själv be om det.

Vill du kryptera större mängder data, eller hela den primära hårddisken, fordras mer drastiska åtgärder. Stora organisationer vill ofta fullständigt kryptera hårddisken på bärbara datorer, så att utomstående inte alls kan komma in i dem. Vidare vill en stor organisation att krypteringen skall ske med en nyckel som företaget har tillgång till, och därmed kan använda för att låsa upp hårddisken om t.ex. den som använder datorn i sitt arbete skulle råka glömma bort sitt lösenord.

Även detta är möjligt. Alla partitioner på en hårddisk kan krypteras genom att kryptera hela den blockenhet som svarar mot hårddisken, t.ex. */dev/hda*. Krypteras den med en förgenererad nyckel istället för med ett lösenord,[18] kan nyckeln lagras undan i organisationens kassaskåp och plockas fram vid behov. Själva nyckeln krypteras sedan med

[18] Detta sker i praktiken helt enkelt genom att skicka nyckeln som lösenord i ett rör till **losetup** och **mount**. T.ex. **losetup -p 0 -e aes /dev/loop0 /var/foo.img < nyckel.bin** respektive **mount -p 0 -o loop,encryption=aes /var/foo.img /mnt/krypto < nyckel.bin**. När det

ett lösenord som är specifikt för användaren (om detta låg i klartext vore det inte mycket mening med det hela).

Operativsystemet kan sedan startas från en mindre hårddisk eller t.ex. en USB-nyckel med ett filsystem motsvarande */boot* där nyckeln ligger i krypterat format. David Braun har skrivit ett utförligt dokument[19] med steg-för-steg anvisningar om hur detta åstadkoms med loop-AES, men principen är exakt densamma för den inbyggda kryperingen i 2.6-kärnan. Att lägga denna typ av kryptering ovanpå en befintlig distribution kan däremot vara lite komplicerat.

Ett alternativ är att använda usermode-baserad kryptering. Från och med version 2.6.14 av Linuxkärnan finns en modul kallas FUSE (utläses *Filesystem in Userspace*) tillgänglig. Denna kan i sin tur användas för att skapa krypterade kataloger och filer som bara används i en liten del av filträdet, och enbart för en viss användare. Till skillnad från helt krypterade hårddiskar kan dessa lagras på nätverksenheter eller andra lagringsmedia som inte krypterats på låg nivå med loopback-gränssnittet.

Det vanligaste FUSE-baserade krypterade filsystemet heter kort och gott *EncFS*[20] och krpyterar alla filer mellan två monteringspunkter. I den ena punkten matar du in okrypterad information, och på den andra punkten dyker den upp ett ett oläsligt format, där varje fil och varje katalognamn är krypterad gallimatias. Den stora fördelen med detta är att du inte behöver kryptera en hel hårddisk eller ha en stor fil skräpandes på din hårddisk, utan att de krypterade filerna allokeras som om de vore vanliga filer och växer dynamiskt. En annan stor fördel är att det är möjligt med inkrementell säkerhetskopiering: även om de krypterade filerna är obegripliga kommer de att säkerhetskopieras som om de vore vanliga filer. När det gäller krypterade loopback-enheter måste du antingen göra en okrypterad säkerhetskopia eller säkerhetskopiera hela blockenheten.

10.5 Intrångsskydd

Intrångsskydd betecknar ett skydd mot att någon utomstående person utan användarkonto på ditt system tar sig in i din dator och använder den till något, eller kopierar något från den.

Det enklaste sättet att ta sig in i ett system på ett stort företag är att

---

gäller den nuvarande versionen av paketet util-linux (version 2.12q i skrivande stund) är nyckeln hårt styrd till 256 bitar med AES och de flesta andra krypteringsmetoder, vilket är detsamma som 32 bytes. Du bör i detta fall generera en binär nyckel med 256 bitar med hjälp av **dd if=/dev/random of=nyckel.bin bs=1 count=32** för att få en bra nyckel.

[19] Se *Disk Encryption HOWTO* [6].

[20] Se: `http://arg0.net/wiki/encfs`

ringa datorsupporten, ”helpdesken”, eller vad den nu kan kallas. Eftersom alla på företaget inte känner varandra, kan ganska begränsad bakgrundskunskap om någon anställd, utkastade referenser till mäktiga personer inom organisationen o.s.v. göra det möjligt att erhålla tillgång till de flesta system bara genom att be om det på rätt vis. Detta tillvägagångssätt går vanligtvis under benämningen *social ingenjörskonst* (engelska: *social engineering*) och skydd mot den typen av angrepp ingår inte inom ramen för denna bok. Vi kommer här att koncentrera oss på tekniska skydd mot intrång.

Det huvudsakliga orosmomentet för POSIX-system på detta område brukar vara säkerhetshål av den typ som gör det möjligt för utomstående att via Internet ta sig in i din dator och agera som root-användaren, ett s.k. root-hål. Ett illasinnat datorprogram som utnyttjar en sådan brist kallas *root exploit* eller *remote root exploit*,[21] för att understryka att det är en för systemet främmande dator som utför angreppet.

En något mindre allvarlig kategori brukar vara säkerhetshål som gör det möjligt för användare på systemet (d.v.s. användare som redan har ett ”vanligt” användarkonto) att förvandla sig till root. Det senare kallas lokalt root-hål och utnyttjas följdaktligen av ett *local root exploit*.

Det första och viktigaste skyddet mot alla typer av root exploits är att uppdatera systemet med alla nya programlappar i samma stund som de kommer ut. Denna typ av svagheter blir snabbt kända av de som av någon anledning håller på med datorintrång av det här slaget.

Om du blir måltavla för ett angrepp av detta slag utan att säkerhetshålet ännu är allmänt känt, kommer du inte att kunna skydda dig med mindre än att datorn är avstängd eller på annat sätt otillgänglig för angriparen (se nedan om brandväggar).

Om du blir måltavla för ett angrepp från militär underrättelsetjänst kommer du inte att märka det. Dessa organisationer plockar systematiskt fram svagheter i system och behåller denna kunskap för sig själva. De är också experter på att dölja sina spår på alla vis, så att ett angrepp inte är synligt i efterhand. Angrepp kommer också att ske på ett koordinerat vis från flera platser på Internet samtidigt.

Är din information så känslig, att detta hot känns obehagligt (är aluminiumhatten på?) är enda utvägen att inte ansluta systemet till Internet över huvud taget, samt strängeligen förbjuda alla som använder systemet att ansluta det till Internet. Normalt kommer du att komma fram till att det är viktigare för din verksamhet att vara ansluten till Internet.

Den vanligaste (tekniska) metoden för en utomstående att på detta

[21] Det finns inga bra svenska ord för dessa problem. Ordet *exploit* betyder *utnyttja* eller *begagna sig av*. Den som begår intrånget *utnyttjar* alltså svagheten, säkerhetshålet.

vis ta sig in i ett system är att avlyssna och stjäla lösenord på nätverket. Osäkra protokoll som FTP, rlogin eller Telnet skickar lösenord i klartext och är också de vanligaste angreppspunkterna. Undvik därför helt dessa protokoll och använd istället SSH-varianter. Avlyssningen sker med en s.k. *sniffer* som avlyssnar paket på Ethernetnivå, vilket är möjligt med alla Ethernetkort som är anslutna till ett nätverk.[22]

Nyare sniffer-program kan även avlyssna trådlösa nätverk, och den som gör det behöver således inte ens ha tillgång till en dator i lokalen.

Därefter är den vanligaste metoden är att angripa systemets demoner med s.k. *buffertöverskrivning* (engelska: *buffer overflow*, *buffer overrun*). Detta innebär att angripare lokaliserar en illa skriven kodbit i ett demonprogram, där en buffert (i praktiken en rad bytes i minnet), som lagrar något som kommer in via nätverket, kan växa ohämmat utan att programmet klipper av tillströmningen. Bufferten, som i allmänhet är temporärt allokerad på programmets stack, kan då skrivas över så långt att den expanderar upp i returadressdelen av aktiveringsposten för rutinen som skapat bufferten. Väl där kan angriparen byta återhoppsvektorn mot en pekare till ett eget maskinkodsprogram som t.ex. öppnar ett Telnetskal. På detta vis kan en bakdörr in i systemet öppnas.[23]

10.5.1 Intrångsdetekteringssystem

Intrångsdetekteringssystem har, som namnet antyder, till uppgift att detektera om du inte lyckats hålla ditt system tätt, om det redan har gått så långt att någon har tagit sig in i ditt system och utfört en s.k. root exploit.

Det första en sådan inkräktare brukar göra när denne väl lyckats bli root på ett system är att installera ett s.k. *rootkit*. Detta innebär att vanliga binärer som **ps**, **ls** o.s.v. byts ut, så att dessa inte kan användas för att avslöja inkräktaren. T.ex. är **ps** modifierat så att du inte kan se de processer som inkräktaren har startat, och allt verkar vara normalt i systemet. **ifconfig** modifieras också så att du inte ser att nätverkskortet har satts i promiskuöst läge och används för att ”sniffa” nätverket efter nya lösenord. Alla upptänkliga metoder för att dölja sig används.

Ett enkelt sätt att leta efter korkade inkräktare är **chkrootkit**, ett skript som kontrollerar om något välkänt rootkit — det finns några stycken — har installerats på systemet.[24] En förslagen inkräktare använder

[22]Bland dessa stygga program finns *Snort*, *Ethercap*, *TCPdump* eller *Ettercap*. Det sistnämnda är nog det som i dagsläget är elakast.

[23]Jag kräver inte att alla läsare skall förstå dessa mekanismer i detalj. En *aktiveringspost* är ett minnesområde som subrutiner i procedurella språk reserverar vid anrop för att lagra rutinens återhoppsadress och lokala variabler.

[24]Se: `http://www.chkrootkit.org/`

i och för sig kanske ett nyare rootkit som vet hur det skall bete sig för att gömma sig även för **chkrootkit**.

Tripwire[25] är ett lite listigare program som gör kontrollsummor på program med jämna mellanrum, sparar undan kontrollsummorna, och kollar dem igen senare. Om kontrollsummorna har ändrat sig har programmet bytts ut, och var det inte meningen att det skulle bytas, så är det sannolikt fråga om ett rootkit. Inte heller detta är vattentätt, men fångar ytterligare några, inte fullt lika korkade inkräktare.

Det finns inkräktare som kan ta sig förbi alla dessa hinder och fortfarande använda systemen som om de vore deras egna. Men de är för smarta för att vi på ett enkelt sätt skall kunna hantera dem. Enda motmedlet är att vara smartare, och det är naturligtvis ett heltidsjobb.

10.6 Brandvägg med netfilter

Officiell hemsida: `http://www.netfilter.org/`

Brandväggar har till uppgift att avskärma ett lokalt nätverk eller en enskild dator från potentiellt skadlig trafik. Själva namnet kommer från de tjocka innerväggar utan fönster som byggs i många hus för att förhindra att en brand sprider sig förbi väggen.

I praktiken hanterar de brandväggar vi skall avhandla här inte bränder utan internetprotokollets IP-paket, och tittar enbart på paketets kuvert för att avgöra om det skall släppas igenom brandväggen eller kastas. Det finns alltså fönster i denna brandvägg, och ordet är kanske inte speciellt väl valt. ”Gränspostering” hade kanske varit lämpligare, eftersom det ger en mer korrekt mental bild av vad en brandvägg egentligen gör: granskar inkommande trafik och släpper bara in det som är godkänt, som s.a.s. har ”rätt stämplar”.

I vanliga fall är brandväggen en speciell dator, som har till uppgift att vara brandvägg och inget annat. Den kan antingen ha en del av Internets adressrymd på båda sidor om sig, eller *maskera* ett nätverk med egna adresser mot nätverket utåt, så att trafiken ser ut att komma från en och samma dator, vilket är ämnet för nästa avsnitt. Brandväggar kan också konfigureras för en enda, enskild dator. Detta kallas ibland för *personlig brandvägg* och det är den typ av brandväggar vi skall titta närmare på i detta avsnitt. (Du kan naturligtvis använda båda typerna samtidigt om du vill.)

Denna typ av funktionalitet har funnits för Linux sedan år 1994 då Alan Cox portade programmet *ipfw* som användes i BSD. År 1998 tog

[25] Se: `http://www.tripwire.org/`

Paul Russell initiativ till att flytta dessa funktioner in i själva Linuxkärnan, och resultatet blev en mycket snabb och dynamisk brandväggsfunktion som kunde kontrolleras med verktyget **ipchains**. I samband med att version 2.4 av kärnan skrevs år 1999 ändrades denna funktion i kärnan radikalt, och ett nytt verktyg med namnet **iptables** skapades för att kontrollera brandväggsfunktioner. Paul Russells projekt att integrera brandväggen i kärnan går under namnet *Netfilter*, men detta är alltså inte namnet på något verktyg, utan bara ett projektnamn.[26]

Innan du börjar filtrera paket beroende på portar kan det vara värt att undersöka vilka portar som egentligen är öppna på din dator. Till detta använder de allra flesta programmet **nmap** som medföljer de flesta distributioner, eller i vart fall enkelt kan laddas ned och installeras.[27]

Programmet har en mängd parametrar men bara **nmap foo.bar.org** säger en hel del om vilka portar som är öppna utåt på en viss dator. Gör inte detta mot datorer du inte själv har ansvaret för: det anses ganska ofint att ”portscanna” folks datorer, ungefär i paritet med att gå omkring någons hus och känna på dörrar och fönster för att se om det är möjligt att ta sig in någonstans. De flesta internetleverantörer förbjuder dessutom sina kunder att syssla med portscanning.

Vi kommer här inte att gå igenom det gamla verktyget **ipchains** utan koncentrerar oss på det nyare **iptables**. De flesta distributioner innehåller funktionalitet för att på ett enkelt och bekvämt sätt konfigurera brandväggsinställningar under installationen. De innehåller ibland också ett verktyg för att editera inställningarna grafiskt.[28] I Debian GNU/Linux och många andra distributioner är programmet **firestarter**[29] populärt för att konfigurera brandväggar grafiskt. Det hjälper även till med att installera startup-filerna så att brandväggen startas automatiskt när du startar din dator.

När ett paket kommer in i Linuxkärnan och modulen med namnet *ip_tables* är inladdad[30] kommer paketen att behandlas enligt olika regler beroende på vart de är på väg. Flödet illustreras i figur 10.4.

Netfilter bygger på att olika *regler* tillämpas på alla paket som kom-

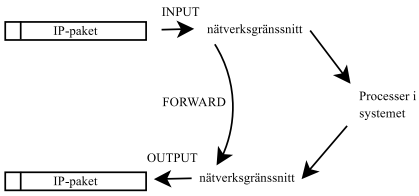
[26]Någon anser kanske att en sådan sak som brandväggsfunktioner inte har i operativsystemets kärna att göra, och det är en helt förståelig synpunkt. Det är nu så fint arrangerat, att det är möjligt att helt avlägsna Netfilter från kärnan genom att välja att inte kompilera den modulen.

[27]Du hittar **nmap** på `http://www.insecure.org/nmap/`.

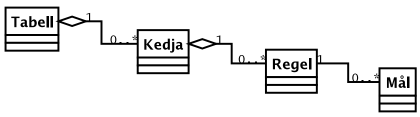
[28]I Red Hat och Fedora Core finns programmet **lokkit**, undangömt i */usr/sbin/lokkit* som sköter detta på ett ganska grovkorningt vis. Lokkit lägger i sin tur alla sina regler i */etc/sysconfig/iptables*. Samma funktion finns även i installationsprogrammet Anaconda.

[29]Se: `http://firestarter.sourceforge.net/`

[30]Linuxkärnans *Netfilter*-del är uppdelad i flera undermoduler, men *ip_tables* är huvudmodulen. Om undermodulerna behövs kommer Netfilter att ladda in dem genom att anropa **modprobe** då vissa regler läggs till.



Figur 10.4: Paketflödet i *Netfilters* filtertabell: paket släpps inte fram till gränssnittet (t.ex. *eth0* eller *ppp0*) om de inte är godkända för *INPUT*. De släpps inte heller ut från systemet gränssnitt, om de inte är godkända för *OUTPUT*. Den tredje regeln, *FORWARD*, används för att skicka paket från ett gränssnitt till ett annat utan att de över huvud taget kommer in i systemet på annat vis. Det senare används för s.k. *nätverksöversättning*.



Figur 10.5: Figuren visar i UML-notation hur de olika objekten i Netfilter relaterar till varandra: *tabeller* är uppbyggda av *kedjor*, som är uppbyggda av *regler* som kan (brukar) innehålla *mål*.

mer in i, lämnar, eller passerar kärnan. En rad regler av en viss typ kallas för en *kedja*, eftersom regler av samma typ tillämpas i tur och ordning, så som länkar i en kedja. En uppsättning kedjor av en viss typ samlas sedan i sin tur i en *tabell*.

Det finns flera tabeller i Netfilter, men den som används för brandväggar heter bara *filter* och om du inte anger något specifikt till kommandot **iptables** så kommer det att anta att det är denna tabell du vill påverka.[31] *Filter*-tabellen innehåller tre typer av kedjor, som alla kan konfigureras med kommandot **iptables**:

INPUT är en kedja av regler som styr vilka paket som får *komma in* i ditt system. Processerna i din dator kommer inte ens att upptäcka att det har kommit ett paket om du inte har en lämplig INPUT-kedjeregel som släpper fram det.

OUTPUT är en kedja av regler som styr vilka paket som får *skickas från* ditt system. Processer som skickar iväg paket som är förbjudna kommer inte att betjänas.

[31]Namnen på alla tabeller i systemet kan skådas med kommandot **cat /proc/net/ip_tables_names** om du har modulen för Netfilter inladdad.

FORWARD är en kedja av regler som bearbetar trafik som skickas direkt från ett nätverksgränssnitt till ett annat. Detta används för nätverksöversättning vilket vi kommer att avhandla i nästa avsnitt. I *filter*-tabellen har denna kedja bara till uppgift att begränsa vad som över huvud taget släpps in till de översatta adresserna.

Ett enskilt paket som skickas till eller från systemet passerar bara *en* av dessa tre kedjor, och kommer alltså bara att behandlas enligt regler av *en* viss typ. Reglerna i sin tur specificeras efter IP-nummer och/eller protokoll, och innehåller så gott som alltid ett *mål*. När ett IP-paket matchar en viss regel, som har ett mål angett, kommer paketet att skickas till det målet. Regler som följer *efter* en regel, mot vilken ett paket har matchat, kommer *inte* att kontrolleras när ett IP-paket väl har skickats till ett visst mål. Följande mål är fördefinierade i Netfilter:

ACCEPT medför att paketet accepteras och släpps in. Detta är det vanligaste målet för en regel: om paketet t.ex. kommer från en viss dator eller en viss grupp av datorer, *skall* det släppas in.

DROP medför att paketet helt sonika kastas, glöms, slängs ut i datarymden. Något meddelande till avsändaren om att paketet inte tagits emot skickas inte.

Det finns fler mål än såhär: paketet kan skickas till en egendefinierad kedja, eller till ett speciell process utanför kärnan och diverse andra exotiska saker. En normal användare använder inget annat än ACCEPT och DROP, men vill du experimentera kan **man iptables** ge dig detaljerad information om alla möjligheter som finns. Själva kommandot **iptables** har några typiska användningssätt:[32]

iptables -t <tabell> -L tar en tabell och listar alla dess kedjor och alla regler i dessa kedjor. Skriver du bara **iptables -L** kommer underförstått tabellen *filter*:s kedjor att visas.

iptables -t <tabell> -F (från engelskans *flush*, spola) rensar ut innehållet i en hel tabell, så den blir tom och fin. Om ingen tabell anges kommer återigen *filter* att påverkas. Görs normalt under uppstart.

iptables -t <tabell> -Z (från engelskans *zero*, nollställ) nollställer paketräknare i alla kedjor i en viss tabell. Görs normalt under uppstart. Varje kedjar håller reda på hur många paket som bearbetats på olika vis.

[32] Som så många andra kommandon som bara kan köras av root ligger **iptables** vanligen i */sbin*-katalogen och det kan därför vara nödvändigt att skriva hela sökvägen: **/sbin/iptables**.

iptables -t <**tabell**> **-P** <**kedja**> <**mål**> sätter en standardpolicy: om ingen annan regel har angetts för ett paket som passerat hela kedjan kommer det angivna målet att användas. "Om inget annat angetts i någon regel, skicka i så fall paketet till detta mål".

iptables -A <**KEDJA**> <**regel**> lägger till en regel i en viss kedja, t.ex. INPUT eller OUTPUT om du arbetar med att bygga en lokal brandvägg. Du kan också ange att du vill påverka en viss tabell genom att hänga på t.ex. **-t nat**. Själva reglerna formuleras genom att ge kommandot ännu fler flaggor. Flaggor som används för regler skrivs med små bokstäver, medan flaggor som skall påverka kommandot direkt skrivs med stora bokstäver. Regler innehåller t.ex.:

-p <**protokoll**> anger att bara protokoll av en viss typ skall påverkas av den här regeln, t.ex. **tcp**, **udp** eller **all** för att välja alla protkoll.

-s <**nät**> (av *source*) anger att paket *från* ett visst nät skall påverkas av regeln. Nätverket anges på formen nät/nätmask (klasslös interdomänform, se sidan 308). För att ange hela nätet 192.168.1.0 skriver du t.ex. **-s 192.168.1.0/24**.

-d <**nät**> (av *destination*) anger samma sak som ovan för paket på väg *till* ett visst nät.

-j <**mål**> anger att regeln skall avsluta med att hoppa till ett visst mål, om den matchar i övrigt. Typiska mål är ACCEPT eller DROP.

-i <**gränssnitt**> anger att paketet skall ha *kommit in* på det angivna gränssnittet för att regeln skall matcha, t.ex. **-i eth0** eller **-i ppp0**. Det är också möjligt att ange att ett paket skall ha kommit på vilket gränssnitt som helst *förutom* detta gränssnitt med t.ex. **-i ! eth1**.

-o <**gränssnitt**> anger att paketet skall *vara på väg ut* till det angivna gränssnittet för att regeln skall matcha. I övrigt gäller samma som för ovanstående regelkomponent.

När detta kommando används för att sätta upp en brandvägg görs det naturligtvis i form av ett skript som körs då systemet startar.

I figur 10.6 illustreras hur kedjan INPUT i tabellen *filter* används för att filtrera bort all trafik till port 1-1024 på ett system, vilket ger en hyfsad brandvägg. Betydligt paranoidare filtrering är naturligtvis möjlig, men möjligheterna att skjuta sig själv i foten genom att spärra trafik som egentligen skall släppas in är också stora.

```
#!/bin/sh
# Här hittar vi förhopningsvis programmen som behövs:
export PATH=/sbin:/usr/sbin:/bin:/usr/bin
# Se till så att den här modulen finns
modprobe ip_tables
# Rensa bort alla gamla regler i ''filter''-tabellen
iptables -F
iptables -Z
# Börja med att slänga *allt* som kommer in
iptables -P INPUT DROP
# Tillåt all trafik från det interna nätverket, 192.168.1.0
iptables -A INPUT -s 192.168.1.0/24 -j ACCEPT
# Släpp in SSH oavsett varifrån det kommer.
iptables -A INPUT -s 0/0 -p tcp --dport 22 -j ACCEPT
# Förbindelser som etablerats från mig själv är OK.
iptables -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
```

Figur 10.6: Skript som är avsett att köras som root (t.ex. i */etc/init.d*) och som blockerar all trafik till port 1-1024 förutom SSH och ping, samt trafik från det egna nätet (192.168.1.0). Lägg märke till standardpolicyn *-P ACCEPT* som släpper igenom allt som inte explicit spärras, så att egna tjänster på portar över 1024 släpps in.

10.6.1 Nätverksöversättning: maskerade nät

Nätverksöversättning är det svenska ordet för engelska *network address translation* (NAT), vilket även brukar kallas för *maskerade IP-nät* (engelska: *IP-masquerading*). Principen för detta är att IP-paket skrivs om i en gateway mellan två nätverk, så att paketen ser ut att komma från en annan dator än de egentligen gör. Paketförvrängningen går i båda riktningarna.

Det absolut vanligaste sättet att använda denna teknik är när bara *en* nätverkanslutning och *ett* IP-nummer finns att tillgå, men du vill kunna ansluta flera datorer till en och samma uppkoppling. Flera internetleverantörer har t.ex. som policy att bara dela ut ett IP-nummer per kund. Med nätverksöversättning kan du bygga ett internt nätverk mellan dina datorer, där datorerna har IP-nummer ur någon ledig serie (t.ex. typiskt 192.168.1.x-serien) och låta dem dela på ett enda IP-nummer utåt, mot Internet.

För datorerna inne på ditt huvudsakliga nätverk kommer den dator som representerar nätverket utåt att ”maskera” de övriga datorerna. Inifrån ditt nätverk kommer datorn däremot att uppfattas som en gateway, vilken som helst. Det är även möjligt att låta den maskerande datorn dela ut IP-adresser och DNS-information på det lokala nätet med

hjälp av DHCP. Att använda denna uppställning kräver[33] två separata nätverksgränssnitt i datorn som maskerar nätet, ett som används inåt, och ett som används utåt. Ofta är detta identiskt med två olika fysiska nätverkskort, men det kan också t.ex. vara ett nätverkskort mot det egna nätet, och en PPP-anslutning utåt.

Det spelar ingen roll om den yttre IP-adressen i ett maskerat nät är dynamiskt tilldelad; nätverket kan dynamiskt maskera sig bakom den adress som erhålls vid anslutningen. Ett visst problem kan dock uppstå med DNS-serverkonfiguration, som måste göras lokalt för varje maskin på det interna nätet om inte DHCP används. Om något i DNS-inställningarna ändras när den maskerande datorn ansluts till t.ex. Internet, kan det bli nödvändigt att konfigurera om och starta om datorerna på det maskerade nätet för att ändringarna ska ”ta”, eftersom dessa inställningar bara läses in från DHCP vid uppstart.[34] Det är emellertid ganska ovanligt att DNS-inställningarna ändras på en dynamisk anslutning.

Om du har en gammal dator som du bara vill använda som brandvägg och/eller nätverksöversättare finns det en minimal Linuxdistribution som bara är till för att starta datorn som brandvägg, och som rymd på en diskett så att du inte behöver ha någon hårddisk i datorn. Denna kallas *flopppyfw*.[35]

10.6.2 Annan paketbearbetning

Utöver *filter* och *nat*-tabellerna finns även en tabell med namnet *mangle*. Reglerna i denna tabell kan misshandla IP-paket och ändra innehållet på olika vis, och kan vara roligt för den som vill experimentera med nätverk eller har vissa väldigt specifika tekniska krav. För den som vill ha riktigt roligt finns även möjligheten att definiera egna tabeller och kedjor utöver de som redan finns.

Vill du kan du med separata regler i Netfilter även *logga* t.ex. alla anrop som spärras på ett visst vis, för att ta reda på om du är utsatt för angrepp eller otillåtna portscanningar. Loggar av det slaget på datorer anslutna direkt till Internet blir snabbt fulla, eftersom det är gott om folk som portscannar trots att det är ofint. Om du inte själv är nätverksadministratör skall du räkna med att din nätverksadministratör

[33]Kräver och kräver. Det är naturligtvis möjligt att tilldela ett gränssnitt mer än ett IP-nummer och på så vis mickla med det hela.

[34]Det är möjligt att lösa detta på ett perfekt vis med en vidarebefodrande DNS-server på den maskerande datorn.

[35]Se: `http://www.zelow.no/floppyfw/` Floppyfw har för övrigt också stöd för qADSL om du råkar ha ADSL-anslutning.

```
#!/bin/sh
# Här hittar vi förhopningsvis programmen som behövs:
export PATH=/sbin:/usr/sbin:/bin:/usr/bin
# Vilka gränssnitt och nät är det fråga om?
INTRANET=192.168.1.0/24
INTERNIF=eth0
EXTERNIF=ppp0
# Om någon modul saknas läggs den in här.
modprobe ip_tables
modprobe iptable_nat
# Slå på nätverksöversättning i kärnan
sysctl -w net.ipv4.ip_forward=1
# Tillåt dynamisk adresstilldelning
sysctl -w net.ipv4.ip_dynaddr=1
# Rensa upp i tabellerna
iptables -F
iptables -t nat -F
# Slå på nätverksöversättning för ppp0 i ''nat''-tabellen
iptables -t nat -A POSTROUTING -o $EXTERNIF -j MASQUERADE
# Skicka inte vidare vad som helst, standardpolicy.
iptables -P FORWARD -j DROP
# Men skicka vidare allt från det lokala nätverket som vill utåt
iptables -A FORWARD -i $INTERNIF -o $EXTERNIF \
    -s $INTRANET -j ACCEPT
# Relaterad trafik får passera tillbaka in
iptables -A FORWARD -m state --state NEW -i $INTERNIF -j ACCEPT
iptables -A FORWARD -i $EXTERNIF -o $INTERNIF -m state \
    --state ESTABLISHED,RELATED -j ACCEPT
iptables -A FORWARD -m state --state NEW,INVALID \
    -i $EXTERNIF -j DROP
```

Figur 10.7: Skript som maskerar ett (litet) internt nätverk bakom den adress som tilldelats gränssnittet *ppp0* efter uppringning (detta kan bytas mot t.ex. *eth1* för att använda två Ethernet-kort). Skriptet är avsett för ett mindre hemnätverk där flera datorer anslutna till ett lokalt nät skall dela på en uppringd förbindelse.

portscannar datorer lite då och då för att kolla säkerheten på nätverket. På dina egna nätverk skall dock bara du själv utföra portscanningar.

Den som vill lära sig *allt* om hur Netfilter kan användas rekommenderas att läsa Oskar Andreassons *Iptables Tutorial*[2]. Det som inte står där är inte värt att veta.

10.7 Virus, maskar o.s.v.

När det gäller den typ av säkerhetshot som utgörs av datorvirus, maskar och trojaner, så har GNU/Linuxvärlden varit förbluffande förskonad från dessa. Orsakerna till detta är omstridda: en del menar att POSIX-system i allmänhet har en bättre säkerhetsmodell, andra menar att virusprogrammerare inte klarar av det krångliga POSIX-gränssnittet, andra åter menar att virusprogrammerare söker sig till den populäraste plattformen, vilket de facto är Microsoft Windows.

Datorvirus är av flera slag och är snarast något av en svepande beteckning på alla elaka datorprogram. Den typ som florerade på 1990-talet spred sig via disketter, där de infekterade huvudbootblocket[36] och sedan kröp vidare från användare till användare. Om du drabbas av ett sådant virus är det sannolikt att din GNU/Linux-installation slutar att fungera helt, eftersom LILO och GRUB använder huvudbootblocket på ett annat vis än de virus som brukar sprida sig där. Denna typ av virus är inte vanliga numera, eftersom de flesta flyttar sina filer via Internet och inte via disketter.

Maskar (engelska: *worm*) är datorprogram som tar sig in via nätverksanslutningen och sätter sig fast i datorn som en egen process eller som en parasit på en annan process. Bara ett fåtal maskar har skrivits som drabbat POSIX-system. De stoppas ofta effektivt med en restriktiv brandvägg, men kan ta sig in genom att utnyttja svagheter i serverdemoner, i synnerhet buffertöverskrivning. Mot detta hot hjälper bara en rutin att hålla systemet ordentligt uppdaterat i alla lägen.

När det så gäller *trojaner*, slutligen, är detta program som brukar utge sig för att vara något annat än de är. Den typ som spritt sig mycket på senare år har florerat i form av bilagor till E-post, som utger sig för att vara ”roliga program”, skärmsläckare, viktiga ordbehandlingsdokument etc. De tillämpningsprogram som används för E-post, och som kommer att avhandlas i avsnittet om tillämpningsprogram, ger sällan möjlighet för användare att köra bilagor genom att bara klicka på dem.

De mycket skilda distributionerna och skrivbordsmiljöerna i GNU/Linux-system gör det också svårt för en virusprogrammerare att veta

[36] Se avsnitt 4.1.3 på sidan 138.

hur systemet där viruset skall köras egentligen ser ut, vilket behövs för spridningen. De flesta Windows-system är däremot ganska lika, och det gör det lättare att skriva ett virus som fungerar överallt.

Det finns ingen garanti för att GNU/Linuxsystem alltid kommer att vara säkra för trojaner, och därför bör den uppmärksamme användaren hålla sig à jour med händelser på området.

KAPITEL 11

Tillämpningsprogram

Tillämpningsprogram, eller applikationsprogram (från engelskans *computer application*, tillämpning av en dator) är program som utför något s.k. nyttigt — enligt någon godtycklig definition av nyttighet — med ditt operativsystem.

Tillämpningsprogram kan definieras genom att beskriva vad de är: program avsedda för *alla* slutanvändare, inte för datorexperter. De kan lika gärna definieras genom att beskriva vad de inte är: de är *inte* operativsystem som GNU/Linux, inte heller systemprogram eller demontjänster.

Äldre tiders tillämpningsprogram för POSIX-systemen var de kommandoradsbaserade program som användare kunde köra i ett skal, och där växla mellan dem i form av olika *jobb*.

Tillämpningsprogram som framställs nuförtiden har grafiska användargränssnitt. De är gjorda för att manövreras med mus och inte enbart med tangentbord. De nyare program som presenteras här är ofta mer eller mindre integrerade i någon av skrivbordsmiljöerna GNOME eller KDE.

Generellt kan det sägas att tillämpningsprogrammen för GNU/Linux har mycket gott stöd för svenska språket. Frivilliga översättare saknas inte, så programmen talar nästan alltid svenska från första version. Skrivbordsmiljöerna GNOME och KDE, som är grunden för de flesta grafiska tillämpningsprogram, är fullständigt översatta till svenska.

11.1 Kontorsprogram

Den arkaiska bilden av tillämpningsprogram är kontorsprogrammen. Jag kommer inte här att gå in på hur dessa program fungerar, eller vad de skall användas till: sådan ytterst grundläggande kunskap besitter säkert redan den som läser denna bok.[1]

Den främsta anledningen till att många valde att skaffa en av de tidigaste persondatorerna, Apple II, var möjligheten att köra programmet *VisiCalc* som programmerats av Dan Bricklin och började säljas år 1979. VisiCalc var det första s.k. *kalkylarket* (engelska: *spreadsheet*).

VisiCalc var också det första kontorsprogrammet: det var avsett som en hjälp under bokföring, något datorer i och för sig redan användes till i stor utsträckning, men bara av specialiserade ekonomiavdelningar på stora företag. VisiCalc och Apple II gjorde det möjligt för mindre företag att använda datorer för att sköta sin bokföring. Detta berodde dels på att datorn och programmet var jämförelsevis billiga, och dels på att programmet i sig var lätt att använda.

VisiCalc hade en enorm inverkan på datorindustrin. Det var sannolikt också den direkta framgången bakom IBM:s PC-arkitektur, eftersom många företag köpte in IBM PC bara för att kunna köra Lotus 1-2-3, ett program som funktionsmässigt var en direkt kopia av VisiCalc och ursprungsprodukten för Lotus Software.[2] Senare skapade även Microsoft ett kalkylark i samma stil med namnet Microsoft Excel. Kalkylarket är också synonymt med begreppet ”killer app” (ung. ”supertillämpning”), en branschterm som används för att beteckna program som är perfekta för användare, fyller ett specifikt behov, och därmed blir oerhört framgångsrika.

Det idag kanske viktigaste kontorsprogrammet är dock ordbehandlaren. De första produkter som kallades för ordbehandlare var specialdatorer från kontorsjätten Wang, med speciella tangenter som gjorde dem speciellt lämpade för just att skriva text.[3] Det är dock senare dagars ordbehandlare, såsom Microsoft Word och MacWrite som fått bli sinnebilden för vad en ordbehandlare är.

Vad som är kontorsprogram utöver dessa två grundpelare är omtvistat. Här är några kandidater:

Presentationsprogram är program avsedda att göra enkla presentationer av den typ som tidigare ofta gjordes på stordia (overhead).

[1] Kontorsprogram kallas ibland även ”produktivitetsprogram” (engelska: productivity application), jag vet inte om detta skall tolkas som att alla andra program är oproduktiva.

[2] Lotus Software ägs numera av IBM.

[3] En sådan ”ren ordbehandlare” har t.ex. en tangent där det står *bold* för att generera fetstil text, och diverse andra finesser, som dagens ordbehandlingsprogram bara kan erbjuda via kontrollikoner, menyer och speciella tangentkombinationer.

Typexempel: Microsoft PowerPoint.

Grafiska databaser är mycket enkla, grafiska databashanterare som är gjorda för att kunna användas direkt av en relativ novis. Typexempel är Microsoft Access (Windows) och FileMaker (Apple Macintosh).

Flödesschemaverktyg är verktyg som kan rita boxar, linjer, flödesscheman, enkla elektronikscheman, organisationsscheman o.s.v. Sinnebilden är det numera Microsoft-ägda *Visio*.

Personliga informationshanterare av engelskans *Personal Information Manager* (PIM) borde kanske för tydlighets skull kallas *hanteringsprogram för personlig information*, och innehåller ett epostprogram, en planeringskalender, en att-göra-lista och en adresslista (ibland även kallad ”Rollodex”). Programmen kan ibland kan användas av flera personer samtidigt, och kallas då *grupparbetesverktyg*, efter engelskans *groupware*. Typexempel från den proprietära världen är Lotus Notes, FirstClass och Microsoft Outlook.

Ritprogram av två slag förekommer: dels *vektorbaserade* ritprogram som använder matematiska kurvor, i synnerhet Bézierkurvor, för att beskriva grafik. Typexempel på dessa är Corel Draw! och Adobe Illustrator. Den andra typen är rasterorienterade eller bitmaporienterade rit- och bildbehandlingsprogram, varav Adobe Photoshop antagligen är det vanligaste.

Diagramgenereringsverktyg för stapel- och tårtdiagram, samt redigeringsverktyg för matematiska formler brukar ingå som ett underprogram till de flesta kalkyl- och ordbehandlingsprogram.

Projekthanteringsverktyg är verktyg för planering och schemaläggning av aktiviteter, resursallokering[4] samt uppföljning av samma aktiviteter. Sinnebilden för denna typ av verktyg är Microsoft Project, som använder ett s.k. Gantt-schema för att ge en hyfsad överblick över tids- och aktivitetsflödet i projektet.

Somliga skulle eventuellt säga att program för World Wide Webbläddring har blivit så viktiga att de också hör till kategorin kontorsprogram, eftersom de numera finns på varje kontor och är absolut nödvändiga för verksamheten i många företag.

[4]Somliga projektledare har ibland en tendens att kalla de människor som är inblandade i ett projekt för ”resurser” snarare än ”medarbetare”.

I GNU/Linux finns det ett antal s.k. kontorsprogramssviter som innehåller delprogram för en eller flera av dessa uppgifter. Andra delar täcks av fristående program.

Det har utpekats som ett viktigt kriteria för kontorsprogramssviter att det måste vara lätt att flytta information mellan de olika delprogrammen; som ett absolut minimum måste användaren kunna klippa ut information av vilket slag som helst i ett program och klistra in det i ett annat. T.ex. skall det vara möjligt att klistra in en tabell och ett diagram som gjorts med hjälp av ett kalkylark på en stordiasida i ett presentationsprogram. Sådana tekniker för blanddokument baseras ofta på olika komponentobjektmodeller, som hjälper de olika programmen att använda objekt (tabeller, bilder, ljud etc.) skapade av andra program utan att känna till detaljerna om dem.[5]

Generellt kan sägas att de båda stora skrivbordsmiljöerna GNOME och KDE har varsin uppsättning kontorsprogram: GNOME har en uppsättning program som är skrivna för widgetuppsättningen GTK+ och använder GNOME:s komponentobjektsystem Bonobo för att utbyta information mellan programmen. KDE å sin sida har en uppsättning program baserade på Qt och byter information med hjälp av KParts.

11.1.1 OpenOffice.org

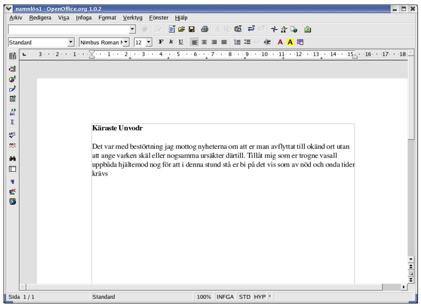
Officiell hemsida: `http://www.openoffice.org/`

OpenOffice.org är utan tvekan den mest använda kontorsprogramssviten i GNU/Linux-världen. Projektet har sitt ursprung i det tyska företaget StarDivision som började utveckla en egen kontorsprogramssvit under namnet StarOffice redan i mitten av 1980-talet. Företaget köptes år 1999 av Sun Microsystems, som i juli året efter deklarerade att källkoden till StarOffice skulle öppnas och bli ett fristående projekt under namnet OpenOffice. (Namnet OpenOffice fick senare ändras till OpenOffice.org eftersom ett annat företag ägde varumärket OpenOffice.)

Programmen i OpenOffice.org-sviten är:

- **OpenOffice.org Writer —** en ordbehandlare
- **OpenOffice.org Calc —** ett kalkylark
- **OpenOffice.org Impress —** ett presentationsprogram
- **OpenOffice.org Draw —** ett vektorbaserat ritprogram som även kan användas som ombrytningsprogram

[5] Det första, mest välkända komponentobjektsystemet för blanddokument (compound documents) var Microsofts OLE, *Object Linking and Embedding*. Numera finns det flera sådana system.



Figur 11.1: OpenOffice.org är den förmodligen mest välutvecklade och använda kontorssviten för GNU/Linux. Bilden föreställer ordbehandlaren *Writer*.

OpenOffice.org är dubbellicensierat: dels är det tillgängligt under GNU GPL, dels är det tillgängligt under en licens som gör det möjligt för Sun Microsystems att modifiera och sälja programmet som proprietär programvara. På det viset utvecklas och säljs numera produkten StarOffice som ett slags derivat av OpenOffice.org. StarOffice tillför en del asiatiska typsnitt, urklippsbilder (”clip-art”), samt den grafiska databashanteraren Adabas. I övrigt är det i princip identiskt med OpenOffice.org.

När Sun tog steget att göra en dubbellicensierad ”öppen källkods”-version av sitt kontorsprogram, kom inspirationen antagligen från Mozilla-projektet. (Vilket vi återkommer till senare i detta kapitel.) Mozilla var vid denna tid också dubbellicensierat och användes i sin tur av Netscape som en bas för den proprietära Netscape-läsaren på ungefär samma sätt som OpenOffice.org numera är grunden för StarOffice.

När StarOffice en gång skapades fanns ingen fungerande skrivbordsmiljö till POSIX-systemen, och därför skapade StarDivision en helt egen skrivbordsmiljö som bara användes till att köra StarOffice. Det hela fungerade som ett enda stort fönster där samtliga program kan köras. Efter Sun Microsystems köp av företaget bröts skrivbordsdelen av programmen bort och OpenOffice.org-programmen körs numera i egna fönster.

Vad som däremot är kvar i programmen är den widgetuppsättning som utvecklats på helt egen väg, och i princip allt som syns i ett OpenOffice-fönster är ritat av programmet självt, inte av t.ex. GTK+ eller Qt.

Vidare har OpenOffice ett helt eget komponentobjektsystem med namnet Universal Network Objects (UNO). Arbete pågår ännu för att integrera OpenOffice.org ordentligt med såväl GTK+ som Qt, så att programmet får det utseende och det användargränssnitt som hör till re-

spektive skrivbordsmiljö.

Filerna som skapas av OpenOffice.org är i XML-format, men är därutöver komprimerade för att inte ta för stor plats. Standardformatet var tidigare ett eget format, men i och med version 2.0 av OpenOffice.org används OpenDocument, vilket är ett format som definierats av en oberoende organisation av kontorsprogramsleverantörer med namnet OASIS. Programmen är även kända för att kunna både läsa och skriva filer från motsvarande Microsoft-program på ett fullt tillfredsställande vis. Om du t.ex. tar emot många dokument skrivna i Microsoft Word, Excel eller PowerPoint, är det normalt inga som helst problem att öppna dem i OpenOffice.org. OpenOffice.org-programmen kan också spara PDF-filer direkt, inga speciella tilläggsprogram behövs för detta.[6]

Det är även möjligt att göra anpassningar i form av makron till OpenOffice.org med ett inbyggt programspråk med namnet StarOffice Basic. Detta språk har stora likheter med Microsofts Visual Basic for Applications, men har ett helt annat programmeringsgränssnitt, så makron till Microsoft Office fungerar inte med OpenOffice.org. Makron kan också spelas in interaktivt.

11.1.2 GNOME Office

GNOME Office är GNOME-projektets kontorssvit. Det innehåller:

- **AbiWord** — en ordbehandlare
- **Gnumeric** — ett kalkylark
- **Dia** — ett flödesschemaverktyg[7]
- **Sodipodi** och **Sketch** — två vektorbaserade ritprogram
- **The GIMP** (*The GNU Image Manipulation Program*) — ett rasterbaserat ritprogram
- **Novell Evolution**[8] — ett kombinerat epost-, planeringskalender- och mötesbokningsprogram
- **MrProject** — ett projektplaneringsprogram utvecklat i Sverige

[6] Detta är en generell egenskap hos POSIX-program: eftersom de flesta program till POSIX-system använder PostScript som utskriftsystem, och eftersom PDF är mycket likt PostScript, är det ofta ganska enkelt för programmen att stödja även PDF. I t.ex. Microsoft Windows används däremot ett helt annat system med namnet GDI, vilket inte har några som helst likheter med PDF, och därför fordras ofta speciell mjukvara för att översätta utskrifter till PDF.

[7] Dia är utvecklat av svensken Alexander Larsson.

[8] Tidigare *Ximian Evolution*.

Utöver dessa program ingår hela OpenOffice.org i princip i GNOME Office, det förklarar varför det är en del ”hål” i programutbudet.

GNOME:s kontorssvit är inte lika tydligt avgränsad mot de övriga programmen i GNOME-familjen, och många som använder GNOME tänker på programmen som en del av GNOME snarare än som en del av kontorssviten.

En speciell särställning intar programmet Evolution (se figur 11.2) som är GNOME:s program för hantering av personlig information. Detta var länge i princip det enda programmet i sitt slag för POSIX-systemen och används även av många som annars i huvudsak använder KDE som skrivbordsmiljö.

Program av denna typ blir mest effektiva som verktyg i en större organisation, om de används tillsammans med en server som gör det möjligt att utbyta kalendrar, skicka mötesinbjudningar samtidigt som tillgänglighet kontrolleras i användarnas kalendrar, dela på nyhetsgrupper och adressböcker o.s.v. Ett samlingsnamn för ett sådant system med både server och klienter är *grupparbetesprogramvara* (engelska: *groupware*), även kallas *samarbetesprogramvara* (engelska: *collaborative software*) eftersom det är inriktat på samarbete i större eller mindre grupper.

Det första systemet av denna typ som blev allmänt spritt var servern Lotus Domino och klienten Lotus Notes, men även servern Microsoft Exchange och dess klient Microsoft Outlook har vunnit stor acceptans. I Sverige har en grupparbetesprogramvara med namnet FirstClass haft stor framgång.[9]

Evolution kan med hjälp av ett tillägg med namnet *Connector* anslutas till Microsoft Exchange-servar av version 2000 eller senare, vilket uppskattats mycket av företag som använder detta system som en del av sin verksamhet. Avsaknaden av ett fritt serverprogram av samma typ som Microsoft Exchange har lett till att OpenOffice.org har startat ett projekt med namnet OpenGroupware.org för att utveckla en sådan server.[10] Ett annat liknande projekt är KDE:s *Kroupware* (se nästa avsnitt). Det finns flera liknande projekt, men inget av dem har ännu uppnått dominerande ställning.

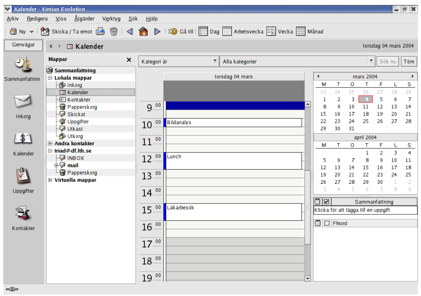
11.1.3 K Office

K Office är KDE-projektets egen kontorsprogramssvit. Det innehåller:

- **KWord** — en ordbehandlare
- **KSpread** — ett kalkylark

[9] Proprietära klienter för såväl FirstClass som Lotus Notes existerar för GNU/Linux.

[10] Se `http://www.opengroupware.org/`



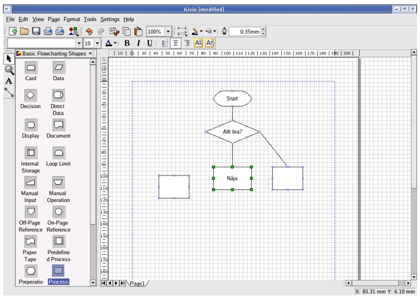
Figur 11.2: *Novell Evolution* är ett personligt informationsprogram (PIM-program) som ingår i GNOME Office.

- **KPresenter** — ett presentationsprogram
- **Kivio** — ett flödesschemaverktyg
- **Karbon14** — ett vektorbaserat ritprogram
- **KRita** — ett rasterbaserat (bitmaporienterat) ritprogram
- **Kugar** — ett rapportgenereringsprogram
- **KChart** — ett diagramgenereringsverktyg
- **KFormula** — en editor för matematiska formler

Ordbehandlingsprogrammet KWord är speciellt: det bygger på ramar i samma anda som Adobes FrameMaker, och inte på flödande sidor som är vanligt i andra ordbehandlingsprogram. Det är således något av ett ombrytningsprogram, om än inte fullödigt. Till skillnad från OpenOffice.org har det inga möjligheter till makrodefinitioner, och är inte speciellt bra på att importera dokument skrivna i Microsoft Word. Att *exportera* dokument i Wordformat går inte alls.[11]

Programmen i KOffice, liksom de andra KDE-programmen, fungerar bra tillsammans och använder samma komponentobjektteknologi (KParts). Att plocka in flödesscheman från Kivio och diagram från KChart i KWord är inga som helst problem.

[11]Läsaren frågar sig säkert varför det är så intressant huruvida ett program kan läsa dokument skrivna i Microsoft Word eller inte — Microsoft Word är ju å sin sida oanvändbart för att läsa dokument skrivna i de olika GNU/Linux-programmen. Anledningen är att många är tvungna att kunna använda detta format p.g.a. dess allmänna utbredning. Naturligtvis är det också viktigt att utbilda användare i att undvika att använda dokumentformat som inte kan läsas av alla.



Figur 11.3: *Kivio* är ett utmärkt flödesschemaverktyg som ingår i KOffice.

KDE har tidigare inte haft något speciellt program för att hantera personlig information. Numera utgör KPIM ett samlingsnamn för ett flertal program för detta ändamål: *KMail*, *KAddressbook*, *KOrganizer*, o.s.v. som sedan sätts samman i en klient med namnet *Kolab Client*. Denna klient arbetar i sin tur mot en serverprogramvara som kallas *Kolab Server*.[12] Projektet som helhet kallas *Kroupware*.[13]

Arbete pågår även på ett grafiskt databashanteringsprogram med namnet *Kexi*.

11.2 World Wide Web-bläddring

Hypertexten, webbservern och webbläsaren uppfanns i ett slag år 1989 då Tim Berners-Lee började arbeta med sitt hypertextkoncept vid CERN i Schweiz. Året efter lade han fram ett projektförslag om att ett hypertextsystem skulle byggas med syfte att förenkla tillgången till olika vetenskapliga artiklar. Det första webbläsarprogrammet hette just *WorldWideWeb*, men bytte senare namn till Nexus, förmodligen eftersom det utvecklades och kördes enbart på NeXT-datorer.

Idén med hypertext var inte ny — andra liknande system som Hypercard och Gopher fanns redan och användes också till viss del, men en bidragande orsak till att de inte blev framgångsrika kan ha varit att de byggde på proprietära protokoll som måste licensieras före användning.

Berners-Lee baserade sitt hypertextsystem på en delmängd av den allmänt tillgängliga dokumentstandarden SGML (Standard General Markup

[12]Kolab Server är i sin tur byggt på andra programvaror som webbservern Apache och databashanteraren Berkeley DB.

[13]Se: `http://kroupware.kde.org/`

Language) och kallade resultatet för HTML — Hypertext Markup Language.

Först runt 1992 lades möjligheten att använda bilder blandade med text på webbsidor in i läsaren. Innan dess var webben helt textbaserad. Detta brukar pekas ut som den centrala innovation som Berners-Lee utförde: efter detta gick det plötsligt att *se* datornätverket, på ett vis som inte bara tillfredsställde de som var vana vid att läsa ren råtext. På så vis hade komponenterna multimedia (text, bilder, ljud, animationer) och globala datornät kombinerats för första gången.

Pei-Yuan Wei vid Berkeleyuniversitetet i Kalifornien uppskattade uppfinningen så mycket att han började utveckla en egen webbläsare med namnet *ViolaWWW*, som bland annat innehöll ett skriptspråk och möjligheter att separera hypertextinnehållet och layoutinformation genom att använda s.k. ”Style Sheets”.

Flera personer vid det amerikanska institutet för superdatortillämpningar i Illinois, NCSA[14], uppmärksammade i sin tur Pei-Yuans program och började utveckla en egen webbläsare med namnet *Mosaic*, och en webbserver med namnet *httpd*. Version 1.0 av Mosaic släpptes den 22 april 1993 och fanns då bara i olika varianter till fönstersystemet *X*. Sent 1993 hade även versioner för Macintosh och Microsoft Windows utvecklats.

En av utvecklarna bakom Mosaic var en sistaårsstudent vid namn Marc Andreessen. Denne träffade genom en slump en av grundarna av Silicon Graphics, Jim Clark, som ansåg att webbläsare hade en kommersiell potential, och startade Mosaic Communications i Mountain View, Silicon Valley. Företaget bytte snart namn till Netscape Communications. Företaget utvecklade i raketfart en egen, proprietär webbläsare med namnet Netscape Navigator som snart blev betydligt populärare än Mosaic. Netscape Communications sålde sin webbläsare ganska billigt, men äldre varianter och betaversioner kunde också laddas ned gratis från företagets hemsida, och detta dög för de flesta användare. Huvuddelen av Netscapes inkomster kom från de serverprogram som såldes under namnet *Netscape Server* (senare *iPlanet*).

Detta inträffade under mitten av 1990-talet, och Internet började då plötsligt att expandera explosionsartat på grund av den oerhört populära webben. Många användare började också använda andra tjänster: FTP, IRC-chattar, och elektronisk post blev snabbt oerhört populärt. Detta gav enorma inkomster till Netscape, vars produkter sålde som smör i solsken, och ledde till en extremt hög värdering av företaget.

Netscapes oerhörda kommersiella framgång ledde till att Microsoft,

[14]NCSA utläses *National Center for Supercomputer Applications*.

som till en början avfärdat hela Internetkonceptet,[15] lade om sin strategi och valde att införliva webbvisionen i Microsoft Windows. För att detta skulle gå snabbt licensierade de källkoden till NCSA Mosaic från företaget Spyglass Technologies, som övertagit rättigheterna från NCSA. Med utgångspunkt från denna mjukvara utvecklades sedan webbläsaren Microsoft Internet Explorer, som efter att den inkluderades som standardkomponent i Windows 95 snabbt växte sig så stor att den fullständigt trängde ut Netscape från marknaden.

Det finns en hel uppsjö av webbläsare till GNU/Linux, men här kommer bara de mest populära, fullgrafiska att presenteras. Om du föredrar ett helt textbaserat gränssnitt finns såväl **lynx** som **links**, av vilka minst en brukar medfölja varje distribution.

Ett tyvärr alltför vanligt problem med webbläsare för GNU/Linux är att webbsajter har utformats för att bara släppa in webbläsare av en viss typ, för ett visst operativsystem. Detta är strikt sett ett missbruk av möjligheten att detektera vilken webbläsare besökaren använder och sajter som gör sig skyldiga till detta bör upplysas om saken.

En lösning som ibland används för att gå runt är att ”maskera” webbläsaren, så att den utger sig för att vara t.ex. Microsoft Internet Explorer. På så vis kan innehållet ändå kommas åt, men samtidigt sänder detta ut en tveksam signal: de som administrerar sajten blir då ännu mer övertygade om att alla i hela världen faktiskt använder Internet Explorer. Använd därför denna möjlighet med omdöme, och underrätta alltid administratörer av problematiska webbsajter att deras sidor inte fungerar i din webbläsare, och att detta är ett tekniskt fel.[16]

11.2.1 Mozilla och Firefox

Mozilla var ursprungligen det interna kodnamn som Netscape Communications använde för produkten Netscape Navigator. Namnet är ursprungligen en sammanslagning av *Mosaic-killer Godzilla*.[17] Namnet blev mer allmänt känt i februari 1998, då Netscape valde att släppa koden för Mozilla / Netscape Navigator under en licens som kallades Netscape Public License som innebar att källkoden publicerades, även om Mozillas status som fri programvara ännu var oklar.

Den stora stötestenen var frågan huruvida licensen var ”kompatibel” med GNU GPL, och kunde användas i GNU/Linux-distributioner.

[15] Microsoft Windows hade under början av 1990-talet inte ens stöd för TCP/IP, utan detta fick laddas ner och installeras separat, eller tillsammans med webbläsaren.

[16] På webbsidan *webbplatskritik* (`http://www.autark.se/webbplatskritik.html`) finns ett forum för vidare debatt i detta ämne.

[17] Godzilla är en enorm radioaktiv ödla som förkommer i en oändlig räcka skräckfilmer från Japan.

Dessa diskussioner ledde till att licensen ändrades till en trippellicens: Netscape Public License, GNU GPL och GNU LGPL, vilket definitivt gjorde Mozilla till helt fri mjukvara.

I samband med detta satte Netscape även upp en separat utvecklingsorganisation som skulle sköta utvecklingen av Mozilla och med jämna mellanrum plocka frukterna av projektet för att producera den proprietära webbläsaren Navigator. Tanken var att detta skulle attrahera utvecklare som arbetade med fri mjukvara och snabbt öka utvecklingstakten på såväl Mozilla som på Netscapes produkter.

Snart efter att källkoden publicerats visade det sig att denna var fruktansvärt rörig och svårbegriplig, och projektet beslutade att helt skrota all kod och skriva om programmet från början. Detta ledde till ett helt oväntat extraarbete, och Netscape kunde inte börja använda koden från Mozilla förrän i december år 2000. Vid det laget hade Mozilla ännu inte ens mognat till version 1.0.

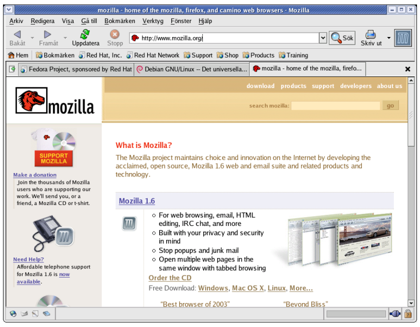
I juli 2003 meddelade den dåvarande ägaren av Mozillaprojektet, America OnLine (AOL) att utvecklingen av webbläsare skulle läggas ned. Projektet överlämnades därefter till en stiftelse, som sedan dess driver projektet.

Under 2004 började två utvecklare, Dave Hyatt och Blake Ross, att skapa en nedskalad variant av Mozilla under namnet Phoenix, eftersom läsaren kommit att bli allt större och oöverskådligare och innehöll för många komponenter som en normal användare aldrig hade glädje av. Mozilla innehöll då:

- Navigator-komponenten, d.v.s. webbläsaren
- Epost- och USENET News-komponenten
- HTML-redigeringskomponenten
- Adressbokskomponenten

Denna blandning av program återspeglar att Netscape tidigare försökte utöka området för sin verksamhet genom att införa allt fler funktioner i Netscape och skapa en hel programsvit under konceptnamnet *Netscape Communicator*. Det är ganska lätt att se att Mozilla i någon mån, utöver att vara en webbläsare, tycks vilja bli ett fullständigt programpaket för hantering av personlig information, i stil med Ximian Evolution. De flesta tänkte trots detta på Mozilla som en webbläsare och inget annat.

I mars 2005 bekräftade Mozillastiftelsen att den tidigare navigatorn skulle brytas isär: webbläsaren som tidigare kallats Phoenix blev en renodlad, snabb läsare under namnet *Firefox* och epost- adressboks- och



Figur 11.4: Mozilla brukar användas till all form av webbsurfande. Här visas de ”flikar” som gör det möjligt att besöka flera webbsajter från ett och samma Mozillafönster.

news-programmet blev fristående under namnet *Thunderbird*. HTML-redigeringskomponenten övergavs helt. Den gamla integrerade Mozilla-applikationen fortlever dock under namnet *SeaMonkey* men utvecklas bara av och för de som absolut måste ha just denna stora kombinationslösning.

Tekniskt sett har Mozilla och Firefox överraskat med mycket hög standard. Projektet är i princip det enda som implementerat de mer exotiska delarna av de standarder som annonserats på löpande band av W3.org.

Firefox har rönt stor framgång sedan våren 2005 och har revitaliserat konkurrensen mellan webbläsare på ett vis som nog få trodde var möjligt. Ett tag innehöll Firefox en funktion som inga andra läsare hade — *flikar* (även kallade ”tabbar”) där användaren med **Ctrl+t** kan öppna en extra ”flik” i webbläsaren, istället för att behöva ha flera fönster öppna på skärmen, för att kunna surfa på flera olika sidor parallellt. Numera är detta standard i alla webbläsare värda namnet. Firefox har ett generellt system för att enkelt installera programtillägg direkt via webbläsaren, vilket kallas XPInstall eller bara XPI. Dessa program är tänkta att installeras i användarens hemkatalog om denne inte är root.

Säkerhetsmodulen i Firefox är omfattande och kan till och med utnyttja NTLM-autentisering[18] vilket kan vara nödvändigt i nätverk där webbservrar av typen Internet Information Server används mycket.

[18]NTLM, NT LAN Manager, är en egentillverkad autentiseringsteknik som utvecklats av Microsoft och som ofta används som standard för att verifiera en användare i Windowsnätverk. Speciellt används denna funktion för att begränsa spridningen av intern information inom stora företag.

Popup-blockerare[19] är inbyggt i Firefox, och kan lätt slås på och användas för att göra surfandet drägligt. Det är också möjligt att ange för enskilda webbsajter om de skall tillåtas att öppna fönster eller inte. Ett populärt XPI-tillägg är **flashblock**[20] som stoppar oönskade Flash-animationer från att dyka upp i webbläsaren.

Firefox har ett säreget konfigurationssystem som innebär att samma konto på en maskin kan användas av flera personer, och varje person kan i sin tur ha flera olika *profiler*. Detta återspeglas i konfigurationsfilerna: i den dolda katalogen *.mozilla* i din hemkatalog finns först en katalog med ditt användarnamn, därefter en katalog med ett helt slumpmässigt namn som är kopplat till varje profil du skapat. I denna katalog ligger sedan alla inställningar, bokmärken o.s.v.

Firefox innehåller ingen lättanvänd funktion för att maskera webbläsaren för att komma åt sidor som utestänger vissa webbläsare. Detta beror på att utvecklarna bakom Firefox inte anser att detta är någon bra taktik för att komma åt problemet. Du kan emellertid maskera Firefox till Internet Explorer genom att gå in i filen *prefs.js* för den profil du använder (eller kanske ännu hellre skapa en separat maskeringsprofil) och skriva till:[21]

```
user_pref("general.useragent.override",
  "Mozilla/5.0 (compatible; MSIE5.5; Windows 98)");
```

Avancerad extrafunktionalitet för Firefox som inte kan installeras via XPI-systemet tillhandahålls genom olika s.k. *insticksprogram* (engelska: *plug-ins*). Dessa skall installeras i katalogen */usr/lib/mozilla/plugins* och inte under */usr/local*-hierarkin, såsom kanske hade varit mest naturligt.

Java med Firefox

Programspråket Java introducerades tidigt som ett språk speciellt lämpat för datorprogram som skulle köras inuti en webbläsare. Skälen var

[19] Popup-fönster är fönster som öppnas av skript på webbsidor du surfar till, och som måste stängas manuellt. Eftersom de hamnar mitt i blickfältet och är svåra att bli av med används de mycket till reklam. De flesta flitiga Internetanvändare utvecklar snabbt en aversion mot detta otyg.

[20] Se: `http://flashblock.mozdev.org/`

[21] Det kanske är någon som tycker att det är underligt att Internet Explorer faktiskt *identifierar sig som Mozilla*! Detta har historiska orsaker: Netscapes läsare, som en gång var dominerande, identifierade sig alltid som ”Mozilla/1.0” o.s.v. När Microsoft skapade Internet Explorer använde de denna sträng för att inte bli utestängda från webbsajter som krävde Netscape Navigator. Nu är rollerna omvända. Internet Explorer identifierar sig som Mozilla, och anger i en parentes att den faktiskt inte är det.

två: dels var språket gjort för att fungera lika bra på alla datorer, dels var det lätt att med Java bygga en högklassig säkerhetsmodell, eftersom språket körs i en s.k. sandlåda.[22] Det är inte möjligt för javaprogram att komma åt delar av systemet som det inte explicit tillåtits att använda. Därför kan Java inte användas för att skapa virus och liknande elaka program.

Javaprogram kan köras som vilka program som helst, men kan också köras med sin grafiska del kopplad till ett fönster i en webbläsare.[23] Ett javaprogram som körs inuti en webbläsare kallas för en *applet*.

Java för Firefox är komplicerat, eftersom den implementation av Java som används mest i GNU/Linux-världen är gjord av Sun Microsystems, och denna är inte fri mjukvara. Det står därför var och en fritt att ladda ned och installera Java från Sun, men det levereras sällan med någon distribution. På senare tid har stor energi lagts ned på utvecklandet av libgcj, vilket är kärnan i GNU-projektets egen javalösning. libgcj kan köra såväl rena javaprogram som javaprogram kompilerade till ren maskinkod, samt kombinationer av båda.

Om Java finns tillgängligt i form av ett XPI-paket är det lätt att installera.[24] Till de flesta distributioner finns dessutom paketsamlingar som glatt distribuerar java i form av RPM- eller DEB-paket.

Om du inte vill installera Java via XPI eller något tredjepartspaket får du först ladda ner och installera Sun:s JRE (Java Runtime Environment). Denna komponent finns i ett flertal former på Sun Microsystems Java-sida.[25] Att installera Java är inte tillräckligt: för att det skall fungera tillsammans med Firefox krävs att ett speciellt insticksprogram installeras.

Att få det insticksprogram för java som medföljer Javainstallationen att fungera är inte alltid så lätt. Den senaste tiden har tanken varit att om din Java-installation ligger i t.ex */usr/java/j2re1.4.1* så finns en insticksmodul i */usr/java/j2re1.4.1/plugin/i386/ns610* som uppenbarligen gjorts för Netscape Navigator 6.10. Denna skall sedan länkas symboliskt till insticksprogramkatalogen för Firefox med något i stil med:

```
ln -s /usr/java/j2re1.4.1/plugin/i386/ns610/libjavaplugin_oji.so
   /usr/lib/mozilla/plugin/libjavaplugin_oji.so
```

Om katalogen */usr/lib/mozilla/plugin* inte finns måste den först skapas.

[22] Java består av interpreterad byte-code, en maskinkod definierad för en processor som inte existerar i sinnevärlden (annat än i några få experimentella versioner).

[23] Den faktiska skillnaden är att programmets huvudklass i första fallet är en subklass till ”application” medan det i en webbläsare är en subklass till ”applet”.

[24] Se: `http://java.mozdev.org/java_xpis/`

[25] Se: `http://java.sun.com/`

Arbete pågår med en fri implementation av Java med namnet *Kaffe*, men denna implementation är i dagsläget inte speciellt bra. Sun och IBM förhandlar dessutom för tillfället om ifall det skulle kunna tänkas vara möjligt att släppa Java som någon form av fri programvara.

Flash med Firefox

Macromedia Flash är ett proprietärt skriptspråk för enkel multimedia i form av bland annat vektorgrafik, som utvecklats som ett lättviktsalternativ till Macromedias multimediaverktyg Shockwave.

Insticksprogrammet för Macromedia Flash är på samma vis som Java ett problematiskt kapitel. Problemet är återigen inte att hitta och installera Flash, problemet är att programmet inte är fritt och inte passar i en GNU/Linux-distribution.

Flash kan laddas ner från Macromedias hemsida och kommer med ett eget installationsprogram som ser till att Flash installerar sig självt. Det är inte svårt att lyckas med detta, om du önskar installera Flash.

Vissa användare upplever det snarast som en fördel att Flash inte fungerar: många annonsörer använder numera ofta stora och iögonenfallande Flashanimationer för att stjäla läsarnas uppmärksamhet, vilket kan uppfattas som ganska störande. Å andra sidan har vissa företag byggt hela system i Flash, och då finns ingen annan utväg än att installera det.

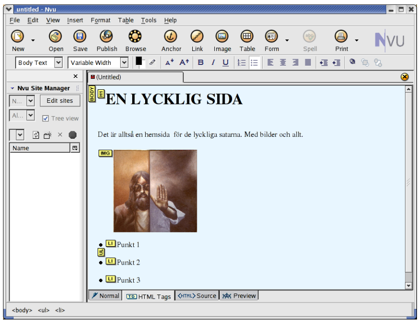
Epiphany

Epiphany är GNOME-projektets tappning av Firefox, som är gjord att fungera speciellt bra i skrivbordsmiljön. I princip består Epiphany av allt utom själva webbfönstret, där hemsidor visas.

Själva visandet av webbsidor sköts nämligen av GtkHTML, en widget som bygger på Firefoxs renderingsmotor Gecko, och Firefox är på detta vis integrerat i GNOME. På andra vis är Firefox inte alls integrerat med GNOME: det har exempelvis ett helt eget komponentobjektsystem, ett eget portabelt plattformsabstraktionsbibliotek o.s.v.

11.2.2 Konqueror

Konqueror är KDE-projektets egen webbläsare. Till skillnad från GNOME:s Epiphany använder Konqueror inga komponenter från Firefox. Konqueror är för övrigt inte bara en webbläsare, utan också KDE-projektets filhanterare. Detta innebär att samma program används för att utforska innehållet i filsystemet på den egna hårddisken som används för att surfa på Internet.



Figur 11.5: Nvu är kanske den mest lovande fullgrafiska webbeditorn.

Den centrala komponenten i Konqueror, KHTML, används även av webbläsaren Safari som medföljer i Apple:s MacOS X och användarbasen är därför mycket stor.

Att maskera Konqueror så att den ser ut att vara en annan webbläsare är mycket enkelt, det kan göras i menyalternativet Inställningar → Anpassa Konqueror → Webbläsaridentifikation. Det är möjligt att anpassa Konqueror så att den uppträder som en annan webbläsare bara mot vissa problematiska sajter. (Kom dock ihåg att ändå upplysa administratören för denna webbsida om problemet.)

Konqueror kan använda insticksprogram för Firefox, så någon separat programvara för detta behövs inte. Om insticksprogrammen installeras i */usr/lib/mozilla/plugins* kommer de att kunna användas även med Konqueror.

11.2.3 Webbredigeringsprogram

Den ursprungliga visionen av World Wide Web var att det skulle vara enkelt för användare att lägga till och redigera artiklar publicerade på webben. Tim Berners-Lee hade för detta ändamål skrivit en robust och enkel editor för datorn NeXT. Eftersom få personer använde denna dator fick programmet ingen större spridning, och de flesta som skrev webbsidor föredrog att göra det genom att editera HTML-koden direkt i någon texteditor. Så länge de som skapade webbsidor var ett fåtal tekniskt kunniga personer fungerade detta ganska bra.

Runt 1995 presenterade företaget Vermeer Technologies den första riktiga fullgrafiska webbeditorn under namnet *FrontPage*. Året efter köptes företaget av Microsoft och produkten blev *Microsoft FrontPage*.

GNU/Linux har länge saknat en motsvarighet till FrontPage, men

det finns ett antal lovande projekt: *Mozilla Composer* följer med Mozilla och duger till de flesta enklare webbsidor, och *Nvu* (vilket förmodligen skall utläsas *envy-you*, "jag avundas dig") är en vidareutveckling av Mozilla Composer som närmar sig FrontPage.[26] SCREEM (Site Creation and Editing Environment) är en webbeditor för GNOME som inte är fullgrafisk, men som är passar bra för avancerade användare.[27]

11.3 Elektronisk post

Användandet av mail, epost, elektronisk post, datorpost eller vad det än kallas, tog sin början i de första stora datorsystemen. Eftersom flera personer kunde använda en och samma dator från flera olika platser och vid olika tidpunkter, var det naturligt att bygga ett system som gjorde det möjligt för dem att utbyta meddelanden med varandra.

När datornätverk blev vanliga föll det sig naturligt att även börja skicka sådana meddelanden mellan olika datorer i nätverket, och därmed var eposten född. I de äldre systemen (baserade på UUCP) var det enda sättet att skicka epost att ange exakt vilka datorer brevet skulle passera på vägen till mottagaren, t.ex. angav *foo!bar!fnord* att ett brev skulle skickas via datorn *foo* till användaren *fnord* på datorn *bar*.

Det moderna användandet av epost bygger på SMTP-protokollet och DNS-systemet. SMTP anger hur eposten skall skickas, och DNS används för att ta reda på en specifik epostdator (MX, Mail eXchange) som pekas ut för varje unik domän. På detta vis kan ett brev till adressen *foo@bar.org* skickas genom att i DNS slå upp den aktuella MX-datorn för domänen *bar.org*, och sedan med hjälp av SMTP överföra brevet till denna dator.

Epostsystemet är ett klient-server-system och delas in i två komponenter:

- *Epostklienterna* som även kallas *mail user agent* (MUA, epostanvändaragenter) och som är den klientdel som de flesta användare använder för att läsa sina brev.
- *Epostservrar* som även kallas *mail transfer agent* (MTA, epostöverföringsagenter) och som har till uppgift att ta emot och vidarebefodra epost till användare på såväl det egna systemet som på andra system på Internet.

Alla POSIX-system är egentligen byggda med tanken att de skall hantera sin egen epost och ha båda dessa komponenter installerade.

[26]Se: `http://www.nvu.com/`
[27]Se `http://www.screem.org/`

(Läs gärna lite om internetservrar för SMTP och IMAP på sidan 423 om du är intresserad av att göra detta.)

mail är en kommandoradsbaserad, mycket enkel epostklient som brukar finnas med i alla distributioner, men den kan bara användas om datorn den används på har konfigurerats så att den självständigt kan sända och ta emot epost via SMTP, d.v.s. om datorn också är en epostserver. Användarnas inkomna post lagras då i katalogen */var/spool/mail* och utgående post brukar hamna i */var/spool/clientmqueue*.

Många användare har numera ingen möjlighet att konfigurera sin dator för att skicka och ta emot epost: dels är det en smula komplicerat, dels är det flera internetleverantörer som helt förbjuder detta och stänger av SMTP-porten (nummer 25) för inkommande trafik till hela nätet.[28] Användarna hänvisas istället till att använda internetleverantörens egna epostservrar, som för det mesta sköts på ett bättre vis. Vissa internetleverantörer öppnar för möjligheten att köra en epostserver på den egna datorn vid förfrågan.

I normalfallet kör du inte någon epostserver på din dator, och har du inte specifikt behov av att göra detta skall du också se till att välja bort detta under installation. Du kan kontrollera om du har en epostserver körande på din dator med **telnet localhost 25**. Om du har en SMTP-server körande kommer den att svara på anropet, i annat fall avbryts försöket.

Många program antar ännu helt kallt att datorn faktiskt *har* konfigurerats för att agera epostserver och skickar därför systemmedelanden med epost i tron att de skall hamna hos den lokala administratören root@localhost. Detta kan göra att katalogen för utgående post i */var/spool/clientmqueue* växer och växer utan att någonsin tömmas om ingen epostserver installerats på datorn. Du kan därför bli tvungen att själv rensa detta utrymme ibland.

Vi kommer nu att anta att du nöjer dig med att kunna läsa och skriva epost, och att någon annan får lov att tillhandahålla din epostserver. Som vanlig användare är det egentligen bara ett fåtal saker du behöver känna till för att kunna få din epostklient att fungera:

- En *epostserver* för *inkommande post*, d.v.s. en dator där din inkommande post landar och mellanlagras. Denna brukar oftast heta något i stil med *mail.foo.org*, *imap.foo.org* o.s.v. Din ”inkorg” på denna dator är ekvivalent mot din fysiska postlåda eller ditt brevinkast.

[28] Anledningen är att sådana system, om de inte sköts ordentligt, är en perfekt relä-central för skräppost. Flera av de Windowsvirus som spritts de senaste åren installerar epostservrar på de datorer de infekterar, epostservrar som i sin tur accepterar att vidarebefodra post från hela världen. Dessa epostservrar upptäcks av skräppostleverantörer, och många internetanvändare har därför omedvetet blivit distributörer för skräppost.

- Vilket *protokoll* din mailserver använder. Vanligast är IMAP, men även det äldre POP3 kan förekomma.
- Ett användarnamn som du kan använda för att ansluta till epostservern.
- Ett *lösenord* som du kan använda för att verifiera din användaridentitet hos epostservern.[29]
- En *epostserver* som kan användas för *utgående post*. I många fall är detta samma dator som servern för inkommande post, men i vissa fall används en separat dator för detta. Den utgående posten använder protokollet SMTP. Datorn för utgående post är ekvivalent mot postens gula brevlådor, som ju också används för utgående post.

Om din server för *inkommande post* stödjer att du använder SSL (Secure Sockets Layer) för att kryptera förbindelsen skall du alltid använda det, och många epostservrar kräver också att SSL används.

En dator för *utgående post* (via SMTP) brukar bara acceptera post från det egna nätverket. Om du använder flera epostkonton samtidigt i ditt epostprogram, kommer de säkert att använda olika servrar för inkommande post, utspridda lite varstans på Internet. Samtliga konton bör däremot använda den server för utgående post som finns närmast dig själv, i ditt lokala nätverk. I annat fall är det stor risk att dina utgående brev aldrig kommer iväg.

Servern för utgående post gör vanligtvis inga försök att verifiera att du verkligen är den person som står på *From*-raden i breven, och vidarebefodrar glatt all post, oavsett vem som skickar den och varthän.[30] Vissa modernare installationer stödjer dock att SSL och lösenord används då även för att skicka post. Stöds detta av både din server och ditt epostprogram skall du naturligtvis använda det.

Några populära epostklienter för GNU/Linux är:

Pine *Program for Internet News and Email*[31] är ett helt textbaserat men mycket välbeprövat och populärt program. Pine använder programbiblioteket *curses* för att generera menyer som kan navigeras

[29]Strikt sett kan din epostserver använda andra metoder för att verifiera att du är den du är. Kerberos är ett sådant system. Dessa alternativa autentiseringsmekanismer är dock ännu inte särskilt vanliga.

[30]Detta är också orsaken till att mycket skräppost skickas genom s.k. ”öppna epostreläer”, d.v.s. SMTP-servrar som accepterar och vidarebefodrar post skickad från alla datorer på hela Internet.

[31]Detta är numera den officiella uttydningen av akronymen. Tidigare betydde Pine *Pine Is Not Elm*, efter Elm (Electronic Mail), en annan epostklient.

med piltangenterna. Naturligtvis kan pine inte visa bilder eller andra multimediabilagor som medföljer vissa brev. Pine är inte fri mjukvara, utan bara gratis, och många distributioner vägrar därför att distribuera Pine. (Pine betyder även ”fura” på engelska.)

Mutt är ett annat textbaserat epostprogram. Detta program är ganska svårt att använda, eftersom det kräver inlärning av en stor mängd tangentkombinationer. Är du van vid sådana program är emellertid Mutt programmet för dig.

Thunderbird är epostprogrammet som utvecklas av Mozillastiftelsen, och som tidigare gick under namnet Mozilla Mail. Detta är mycket populärt och fungerar bra.

Evolution och **Balsa** är GNOME-projektets två epostprogram. Evolution är större och mer tungarbetat, men erbjuder samtidigt gruppарbetesprogrammets alla fördelar med kalender, att-göra-lista o.s.v.

KMail är KDE-projektets epostprogram och ingår som tidigare nämnts i deras personliga informationshanteringssvit.

11.3.1 Skräppostfilter

Alla som använder epost i större utsträckning kommer förr eller senare att drabbas av skräppost, även kallat *spam*. Skräpposten består mestadels av reklam, men kan även vara rena svindlerier eller bedrägeriförsök. En del virus i form av trojanska hästar brukar också kallas skräppost eftersom de tar upp plats och tid för epostanvändare, speciellt gäller detta GNU/Linux eftersom många av dessa virus är Windowsprogram och helt ofarliga för en GNU/Linux-användare.

Flera program har på senare tid utvecklats för att sortera bort och lägga skräppost åt sidan. Många av dessa program är fri mjukvara, t.ex. *SpamAssassin* eller *Bogofilter*. Dessa bygger i allmänhet på statistik över vilken typ av ord som ingår i vissa typer av brev, och måste först tränas på en korpus av existerande skräppost, samt existerande godkänd post.

En annan typ av skräppostspärr använder svartlistning och förbjuder helt vissa epostservrar att ansluta sig till den egna epostservern om det är känt att dessa servrar tillhör kända skräppostavsändare.

Dessa program gör mest nytta om de installeras på din epostserver, och många internetleverantörer använder dem idag regelmässigt för att begränsa skräppostproblemet. Om detta inte är gjort på den epostserver du använder, finns möjligheten att köra den första typen av program lokalt på den egna datorn. Metoden för att göra detta varierar mellan

alla typer av epostklienter och det går därför inte att ge någon generell beskrivning av hur detta skall gå till.

11.4 Chattprogram

Chattprogram (engelska: chat, instant messaging) finns i många former. De är alla mer eller mindre vidareutvecklingar av IRC, (Internet Relay Chat) som för övrigt fortfarande används en hel del.[32] Chattprogram innebär att textmeddelanden som inte är längre än enstaka rader kan överföras mellan användare på Internet, till skillnad från epost, som ju är tänkt att användas för fullständiga brev. Chatt ger istället möjlighet till direkt dialog.

Samtliga chattprogram bygger på klient-server-system där användarna ansluter sina klienter till chattservrar.

Medan IRC bygger på att användarna aktivt ansluter sig till en viss grupp där ett visst ämne diskuteras, är de personliga chattprogrammen, varav urtypen är *Mirabilis ICQ* (I-seek-you) avsedda att vara påslagna hela tiden, och uppmärksammar användaren på om någon annan Internetansluten användare vill kommunicera med dem. Meddelanden kan också lagras, och ibland kan även hela filer skickas till andra användare via chattprogrammet.

Flera företag har tillverkat chattprogram, och många av dem är inte kompatibla med varandra; användarna kan alltså inte kommunicera med andra användare som har installerat ett annat chattprogram.

I GNU/Linux är chattprogrammet *Gaim*[33] näst intill helt dominerande. Det följer också med som standardkomponent i de flesta distributioner. Detta program kan kommunicera med såväl AOL Instant Messager (AIM), ICQ, MSN Messenger, Yahoo Messenger, IRC, Jabber, Gadu-Gadu, och Zephyr.

11.5 Peer-to-peer

Nära besläktade med chattprogrammen är de s.k. *peer-to-peer*-programmen, kort kallade P2P-program. ”Peer-to-peer” blir på svenska närmast like-till-like eller kompis-till-kompis. Dessa program är utformade enbart med syftet att olika internetanvändare skall kunna utbyta filer med varandra. Oftast är det musik och filmer som byts, och sällan är detta förfarande särskilt lagligt.

[32] I detta sammanhang kan det vara lämpligt att tipsa om IRC-servern *irc.freenode.net* där det finns kanaler för att diskutera i princip alla POSIX-system.

[33] Se: `http://gaim.sourceforge.net/`

Urtypen för peer-to-peer-programmen var Napster, ett program som utvecklades av ett företag med samma namn och som sedermera köptes upp och mer eller mindre lades ned. En hel uppsjö med besläktade program har emellertid dykt upp i dess ställe, och det är omöjligt att räkna upp dem alla här.

De flesta peer-to-peer-program av den här typen finns i motsvarande versioner för GNU/Linux, ofta utvecklade mycket snart efter att ett nytt sådant program introducerats. Till exempel finns klienten *valknut* för Direct Connect och en officiell klient för det distribuerade systemet Bittorrent.

11.6 Ombrytningsprogram och typsättning

Ombrytningsprogram, typsättningsprogram eller *desktop publishing*-program (DTP-program) är en typ av program för att utföra professionellt layoutarbete. Under 1960- och 1970-talet ersattes de gamla sättmaskinerna på tryckerier här i landet med speciella datorer som användes för att sätta t.ex. böcker och tidningar.

När Apple Macintosh introducerades uppstod snart en rad program som kunde användas av noviser för enkel typsättning. Bland dessa fanns bl.a. det populära Adobe PageMaker.

En rad professionella program uppstod också: Quark Xpress, RagTime, Adobe InDesign och Adobe FrameMaker hör till denna grupp. Vissa mindre tidningar använder dock även Adobe PageMaker professionellt. De flesta av dessa program körs ännu på Apple Macintosh i professionella sammanhang.

För GNU/Linux finns inget riktigt bra fritt alternativ för ombrytning. Det finns inte ens några proprietära program. Det hade varit lätt för Adobe att göra en utgåva av FrameMaker för Linux, eftersom programmet redan är anpassat för POSIX-systemet Solaris, men de har hittills valt att inte göra detta.

Ett program som ännu utvecklas är *Scribus*, som verkar mycket lovande och ständigt förbättras i hög takt.[34] För enklare arbeten kan eventuellt *KWord* eller *OpenOffice.org Draw* användas.

För matematisk text och annan löpande text, t.ex. vetenskapliga uppsatser, är LaTeX ett bra alternativ för den som orkar lära sig det. LaTeX är Leslie Lamports variant av Donald Knuths program TeX, ett typsättningsprogram som ursprungligen skrevs av Knuth för att typsätta hans egna böcker.

[34]Se: `http://www.scribus.net/`

LaTeX skiljer sig helt från de ombrytningsprogram som nämndes i början genom att det inte är grafiskt, bara resultatet är grafiskt. Själva råtexten och textformateringskommandon skrivs i form av en ren textfil, som i praktiken blir till ett datorprogram som talar om hur texten skall se ut. Texten måste sedan kompileras med LaTeX, och resultatet blir en PostScript- eller PDF-fil. LaTeX är ganska komplext och svårt att lära sig, men om du skriver matematiska eller tekniska artiklar *måste* du lära dig LaTeX, det är helt enkelt standardprogrammet för att skriva sådana artiklar. För att lära sig LaTeX krävs det att du läser en hel bok eller någon av de olika LaTeX-manualer som finns på Internet.

LaTeX kommer i olika distributioner, precis som Linux. Det som skiljer distributionerna är en rad tillägg, ordlistor och liknande. Den vanligaste distributionen för GNU/Linux är teTeX.[35]

Denna bok är skriven med LaTeX.

11.7 Multimedia

Att använda datorn som multimediamaskin blev populärt under slutet av 1990-talet, då flera datorer såldes som ”multimediadatorer”. Detta brukade betyda att datorn hade ljudkort, CD-ROM-spelare och ett par lösa högtalare med förstärkare.[36]

Det fanns program för att spela upp ljud och video mycket tidigt, och det fanns speciella multimediaskivor att köpa med varierande innehåll. Samtidigt pågick utvecklingen av digitalradio (DAB) i ett EU-finansierat projekt med namn EU-147. Projektet innebar en chans för tyskarna Karlheinz Brandenburg och Jürgen Herre att utveckla en del av MPEG-standarden[37] som kallades MPEG Audio Layer 3, senare mest känt som MP3.

MP3-formatet var avsett att användas av digital elektronik, men implementerades även i form av programvara. En sådan programvarukomponent som kan både skapa och spela upp filer innehållande komprimerat ljud eller bild kallas *codec*, efter engelskans *encoder-decoder*. Att en codec för MP3-formatet fanns tillgänglig förändrade inte speciellt mycket.

Den stora förändringen på multimediafronten inträffade 1994-1995. Internettillgången började nu växa explosionsartat. Samtidigt spreds program som kunde komprimera ljudet från CD-skivor till formatet

[35] Själva projektet för LaTeX i sig finns på `http://www.latex-project.org/`.

[36] De flesta är nog ganska ense om att beteckningen ”multimediadator” egentligen betydde ”speldator”.

[37] MPEG, Motion Picture Experts Group, är en liten standardkommité som sysselsätter sig med att skapa standarder för komprimering av ljud och bild.

MP3, och spelarprogram som kunde spela upp dem blev allt vanligare. Tobias Bading skrev då ett program med namnet MAPlay (MPEG Audio Player), som kunde användas för att spela MP3-filer på POSIX-system.

Hösten 1999 hade användningen av MP3 vuxit och blivit mycket utbredd. I detta läge uppstod den mycket populära filbytartjänsten Napster, och med denna följde en uppsjö filbytarprogram av olika slag. Stora skaror internetanvändare började nu byta musik i form av MP3-filer.

Ungefär samtidigt plockade den franska hackaren Jerome Rota ut codec:en för MPEG-4-videokomprimering ur ett Microsoft-program, varefter han kombinerade den med en codec för MP3-ljud och möjligheten att spara resultatet av kompressionen i en s.k. AVI-fil (Audio Video Interleave). Detta "MPEG-4-video plus MP3-ljud" kallade han *DivX ;-)* för att driva med videoformatet DIVX (Digital Video Express) som utformats av ett amerikanskt företag.[38]

På grund av patentproblem rörande MP3-formatet (patenten på formatet ägs av Fraunhofer Institut Erlangen) som gjorde det svårt att på ett tillfredställande sätt använda MP3-filer i fri mjukvara, har ett parallellt komprimerat ljudformat med namnet *Ogg Vorbis* utvecklas. Själva ljudformatet är det som kallas *Vorbis*, medan *Ogg* är namnet på filformatet där bland annat denna typ av komprimerat ljud kan lagras. Detta format har utvecklats med tanken att det skall vara fritt från patenterad teknik.[39] Formatet har ersatt MP3 som standardljudformat i t.ex. Red Hat Linux av detta skäl.

Ett helt förlustfritt format har också utvecklats, under namnet *FLAC* (Free Lossless Audio Codec). Detta används av audiofiler med stor hårddisk som inte vill förlora någon kvalitet från sina CD-skivor: FLAC tar inte bort någon som helst information ur den ursprungliga signalen, bara komprimerar den. FLAC-filer är mindre än ren PCM-data (som används på CD-skivor) men inte *mycket* mindre, bara runt 50% av originalfilstorleken.

Uppspelning av CD- och DVD-skivor, samt uppspelning och kopiering av mediafiler bestående av MP3- eller Ogg Vorbis-filer med musik och DivX-filmer är numera vad gemene man förknippar med multimedia på datorer. Denna situation är inte omtyckt av mediaindustrin av diverse skäl, som vi inte har utrymme att avhandla här.

---

[38] Det "riktiga" DIVX var avsett att göra det möjligt att hyra en film och titta på den i 48 timmar, varefter en förnyelseavgift skulle betalas. Filmen, en form av DVD-skiva, kunde däremot behållas. Projektet blev inte alls lyckat.

[39] Ständiga påpekanden gör gällande att tekniken i Ogg Vorbis "säkert är patenterad" men hittills har ingen kunnat peka på ett patent som visar att det verkligen skulle vara så.

11.7.1 Ljudsystemet i kärnan

Systemet för att visa bild i Linux har redan beskrivits i kapitel 6 om fönstersystemet *X*. Vi skall däremot göra en kort presentation av ljudsystemet.

Ljudet i Linuxkärnan hanteras genom ett blockenhetssystem i */dev*. Till en början fanns det ett fåtal drivrutiner för de allra vanligaste ljudkorten i kärnan, kopplade till */dev/audio*. Efterhand som ljudkorten blev mer avancerade tillkom fler blockenheter och enhetsfiler, och till sist byggde företaget 4Front Technologies en kärnmodul som kunde hantera i princip alla ljudkort på ett likvärdigt vis. Denna fick namnet *Open Sound System*, OSS.

I och med version 2.6 av Linuxkärnan byts OSS helt ut mot ett nytt och bättre ljudsystem med namnet *Advanced Linux Sound Architecture*, kort kallat ALSA. ALSA är till största delen kompatibelt med OSS, och som användare märker du nog bara skillnad när du arbetar med professionella ljudsystem för hårddiskinspelning eller med MIDI.[40]

Rent ljud i form av digitala strömmar går ut till ljudkortet via blockenheten */dev/dsp*. Det kommer sedan att blandas med annat ljud, t.ex. från CD-spelare eller ljudkortets MIDI-syntesenhet (om en sådan finns). Det kan hända att det blir bråk om */dev/dsp* ifall två program samtidigt försöker använda enheten. Av detta skäl har ljudhanterande demoner uppfunnits. Detta är program som har till uppgift att blanda samman digitalt ljud innan det skickas till */dev/dsp*.

Skrivbordsmiljön GNOME har länge använt en sådan ljuddemon som heter **esd**, *Enlightenment Sound Daemon*,[41] medan KDE använder **artsd**, (Analog Real-time Synthesizer Daemon). Ingen av dessa båda demoner är egentligen tillräckligt bra för professionellt bruk, främst därför att båda introducerar någon form av fördröjningar i ljuduppspelningen. Därför har ännu en ljuddemon med namnet **jackd** (JACK Audio Connection Kit)[42] utvecklats.

JACK är i en helt annan klass än de andra demonerna: det kan t.ex. be Linuxkärnan att ge programmet speciell prioritet för att öka ljudgenomströmningen, om kärnan har stöd för lågfördröjningsschemaläggning (realtidsstöd). JACK kan även utnyttja Steinbergs ASIO-protokoll (*Audio Stream Input Output*) som finns implementerat i professionella (dyra) ljudkort. ASIO-protokollet är till för att minimera fördröjningar

[40] MIDI, *Music Instrument Digital Interface* är ett gränssnitt för att ansluta digitala musikinstrument till varandra, eller till datorer.

[41] Denna kommer ursprungligen från fönsterhanteraren Enlightenment ("upplysningen").

[42] Det finns dessutom ett CD-ripparprogram med samma namn, så det kan ibland vara lite förvirrande att försöka hitta Jack. Se: `http://jackit.sourceforge.net/`



Figur 11.6: XMMS är en GTK-baserad mediaspelare, här med ett av de fantasifulla ”skal” som utmärker vissa av dessa program.

i ljuduppspelningen.

JACK kan också skicka ljud mellan olika självständiga tillämpningsprogram och används även för MIDI-signaler. JACK fungerar bara med det nya ljudsystemet ALSA, inte med det gamla OSS, och används mest av program avsedda för ljudeditering eller hårddiskinspelning.

För ditt dagliga bruk av ljudet i GNU/Linux räcker det gott om dina program kan använda OSS eller ALSA direkt utan mellanliggande ljuddemoner.

11.7.2 Mediaspelare

Mediaspelare är program som kan spela upp ljud och video. Program som *MAPlay* och *mpg123* var de första mediaspelarna för GNU/Linux och kördes från kommandoraden. Något enstaka program för att spela upp MPEG-filmer utan ljud fanns också.

När den trevliga mediaspelaren WinAMP (Windows amplifier) blev populär år 1997 ledde det snart till olika försök att åstadkomma likadana program för GNU/Linux. De mest framgångsrika är det svenskutvecklade *XMMS*, *X Multimedia System*[43] och KDE-projektets *Noatun*.[44] Båda spelar såväl MP3- som Ogg Vorbis- och FLAC-filer.

Ingen av de båda spelarna har riktigt alla egenskaper som kan förväntas av en mediaspelare, utan är primärt inriktade på att spela upp ljud, även om de även kan spela vissa videoformat, antingen på egen hand eller med hjälp av olika insticksprogram.

Ett ytterligare ljudinriktat multimediaspelarprogram är GNOME-projektets *RhythmBox*, som är gjort för att efterlikna Apple Computers

[43]Se: `http://www.xmms.org/`

[44]Se: `http://noatun.kde.org/`

iTunes-program. Detta program hjälper till att organisera musiksamlingen och gör det lätt att snabbt hitta den sökta musiken.

Om du vill spela upp *filmer*, såväl från DVD-skivor som från DivX och andra mediafiler, brukar i huvudsak tre olika program användas: *MPlayer*,[45] *Xine*,[46] och *VLC* (Video LAN Client).[47] Det sistnämnda programmet var ursprungligen avsett som klient för TV-utsändningar via Internet, men fungerar även utmärkt som mediaspelare. VLC stödjer även DTS-ljud, något som nog fler program snart kommer att göra.

MPlayer är den spelare som kan hantera flest videoformat, och om den inte själv kan spela upp ett format kan den använda en DLL-fil från Microsoft Windows som "insticksprogram" för att avkoda video. De flesta som använder MPlayer brukar köra programmet direkt från skalet. *Xine* och *VLC* är båda utformade primärt för grafiska användargränssnitt. De olika projekten utbyter stora mängder information, och alla tre använder programbibliotek från projektet FFmpeg för att hantera flera videoformat.

Alla tre programmen kan spela upp DVD-skivor med hjälp av två programbibliotek som heter *libdvdread* och *libdvdcss*. Det första biblioteket är till för att läsa av skivan (utan att montera den som ett filsystem) medan det andra är till för att dekryptera innehållet.[48] Program av denna typ har vållat en del rabalder, men efter att filmindustrins försök att hindra program av denna typ helt har misslyckats, får det hela anses vara överstökat.

Det är även möjligt att använda den proprietära mediaspelaren *Real Player* under GNU/Linux genom att ladda ner den från företagets hemsida. Programmerarna på Real Networks arbetar numera enligt en "öppen källkods"-modell av samma art som Mozilla eller OpenOffice. org med en spelare med namnet *Helix DNA Client*.[49] Helixklienten är dock inte fri mjukvara.

11.7.3 "Rippning"

Med "rippning" av CD- eller DVD-skivor avses att innehållet överförs från skivan till hårddisken. Vissa delar av detta förfarande beskrivs under avsnitt 7.4 om CD-bränning, avseende audio-CD.

---

[45] Se: `http://www.mplayerhq.hu/`

[46] Se: `http://xine.sourceforge.net/`

[47] Se: `http://www.videolan.org/`

[48] libdvdcss dekrypterar skivan genom att prova flera olika metoder: först försöker det att använda en känd kryptonyckel och på så vis imitera en existerande DVD-spelare. Om detta inte fungerar knäcker den helt sonika nyckeln med en av två möjliga metoder.

[49] Se: `https://helixcommunity.org/`

Det finns en mängd program som är avsedda enbart för att kopiera audio-CD och konvertera dem till t.ex. MP3-filer, ett sådant är **Grip**.[50] Detta program använder i sin tur en rad andra program, bl.a. **cdparanoia**, **lame** och **oggenc** för att kopiera CD-ljudet till hårddisk, och därefter komprimera det till MP3, Ogg Vorbis eller FLAC. Grip fungerar så att det kopierar första spåret till hårddisken, och sedan börjar komprimera det första spåret direkt när detta är kopierat, parallellt med att andra spåret läses in, vilket gör att det går snabbt att rippa stora mängder CD-skivor.

Även DVD-skivor kan rippas, i detta fall till t.ex. DivX- eller rent MPEG-format. Ett populärt format är MPEG-1 video, eftersom detta används av Video CD-formatet (VCD), eller MPEG-2 som används i Super VCD, SVCD. Dessa format kan brännas på en eller flera CD-skivor och kan spelas upp i de flesta moderna DVD-spelare, men ger inte mycket högre kvalitet än VHS-kassetter. Ett vanligt program för detta ändamål är **transcode**.[51] som i sin tur används av flera grafiska program.

Att rippa DVD-skivor är ganska komplicerat och möjligen förbjudet. (Rättsläget är oklart.) Den som önskar göra detta kan säkert följa länken till **transcode** och lära sig mer om hur det hela går till.

11.8 Diverse

Här följer en lista av mindre program och de behov de kan fylla. Webbadresser anges inte för programmen, men de har ofta namn som är så unika att du lätt kan hitta dem med en Internetsökning om du skulle vilja. Listan över de populära programmen ändrar sig hela tiden, så det som finns upptaget här kan snart vara inaktuellt. En webbplats som övervakar alla nya program är *freshmeat.net*.

Arkivhanterare är till för att hantera filarkiv av den typ som används för att packa ner program- eller datafilsamlingar, eller för att göra säkerhetskopior av delar av ett filsystem. Det första riktigt populära grafiska arkivhanterarprogrammet var *WinZip*. I GNU/ Linux finns *File Roller*, som är en del av GNOME och *KArchiver* för KDE. Programmen hanterar filarkiv av typen .zip, .tar, .gz, .bz2, .rpm, .lha, .rar o.s.v.

Utvecklarmiljö från engelskans *Integrated Development Environment*, *integrerad utvecklarmiljö*, (IDE) är ett program att programmera i.

[50] Se: `http://grip.sourceforge.net/`

[51] Se vidare på: `http://www.theorie.physik.uni-goettingen.de/~ostreich/transcode/`

Det sköter editering, kompilering och versionshantering av alla filer som ingår i ett programmeringsprojekt. Det är inte alla programmerare som tycker om utvecklarmiljöer, men för dem som gör det finns t.ex. KDE-projektets *KDevelop* eller det alltmer uppmärksammade *Eclipse* att tillgå. (Eclipse är skrivet helt i programspråket Java.)

MIDI-inspelningsprogram är program för att lagra ljud som spelas på keyboard eller elpiano och som anslutits via MIDI-gränssnittet, som finns inbyggt i de flesta professionella eller semi-professionella ljudkort. De kan dels fungera som en "bandspelare" för MIDI-signaler, men är också användbara för att göra t.ex. notutskrifter. Det första programmet av detta slag som kom till allmän användning var Steinbergs *Cubase* för Atari ST (senare även för Macintosh och Microsoft Windows). I Linux är bl.a. programmen *Rosegarden* och *MusE* (Music Editor) avsedda att ge motsvarande funktionalitet. Stödet för att bland annat emulera VST-mjukvarusyntar eller att göra egna, linuxspecifika mjukvarusyntar har ökat allt eftersom programmen har utvecklats. Linuxmotsvarigheten till VSTi (mjukvaruinstrument) heter DSSI (DSSI Soft Synth Instrument).

Ljudediteringsprogram är avsedda för hårddiskinspelning av digitalt ljud, d.v.s. inte för digitala instrument (som med MIDI) utan för rent ljud. Populära program i denna genre är Cubase VST (Virtual Studio Technology) eller CoolEdit för Microsoft Windows. För GNU/Linux finns bl.a. *Audacity* som är lättanvänt och semiprofessionellt, och *Ardour* som är svåranvänt och mycket professionellt. Eftersom all form av ljudeditering ställer speciella krav på Linuxkärnan har speciella musik- och multimediadistributioner utvecklats som är inriktade på ljust sådant.

Genealogiprogram används för släktforskning och hjälper till att bygga en databas över släktingar och ritar släktträd med mera. I Sverige används framför allt program som DISGEN, som bara är tillgängliga för medlemmar i en viss släktforskningsförening. För GNU/Linux finns bl.a. programmet *GRAMPS* (Genealogical Research and Analysis Management Programming System) som även är översatt till svenska.

Datorspel är inte lika vanliga för GNU/Linux som för t.ex. Microsoft Windows. Det finns ett antal vanliga spel som följer med skrivbordsmiljöerna GNOME och KDE, nästan alla är av typen brädspel. Många av dem inkluderar möjligheten att via Internet an-

sluta sig till en spelserver och hitta en slumpvis motpart att spela med. De på senare år så populära *first person shooter*-spelen (Doom, Quake, Unreal o.s.v.) finns ibland för GNU/Linux, men detta är i princip undantag som bekräftar regeln att nya spel till Microsoft Windows *inte* finns till GNU/Linux. Om ditt enda intresse här i livet är att spela nya datorspel kan GNU/Linux *inte* rekommenderas, även om det finns många tricks för att trots allt få dessa program att fungera i GNU/Linux. En populär spelform är emulatorn *MAME* (Multiple Arcade Machine Emulator) som kan emulera de flesta äldre s.k. arkadspel som ännu kan hittas i pizzerior och liknande. Detta program fordrar dock att du har tillgång till programmen (ROM) som fanns i dessa spel.

Situationen för spel till GNU/Linux förändras så sakteliga: allt fler spel dyker upp, och vid en viss punkt kan vi nog räkna med att spel levereras så att de kan köras på både Windows och GNU/Linux.[52]

[52]Det går även att använda Windowsemulatorer för att spela Windowsspel i Linux, se sidan 413.

BILAGA A

Att byta från Windows till GNU/Linux

> Roky Erickson? Vad är det för skit? Det låter ju inte alls som Pet Shop Boys!
>
> — Okänd

Detta avsnitt är avsett att ta upp några av de speciella skillnader som finns mellan de två operativsystemen Microsoft Windows och GNU/Linux. Två saker kommer speciellt fokus att läggas på:

- De som tidigare använt Windows på en avancerad nivå, och snabbt vill kunna göra sig hemmastadda, genom att lokalisera vilka delar av GNU/Linux som svarar mot välbekanta delar av Windows. Har du aldrig använt Windows kan du helt hoppa över denna del av appendixet.
- Att använda GNU/Linux i en blandad miljö där både Windows och GNU/Linux skall existera sida vid sida, och samtidigt kunna utnyttja respektive systems resurser.

Båda delarna kommer att behandlas om vartannat, och går inte helt att skilja åt. För att kunna administrera en blandad miljö måste du t.ex. ha goda kunskaper inom både GNU/Linux- och Windowsadministration.

Liksom analogin i citatet ovan antyder är det i någon mening ganska dumt att jämföra Microsoft Windows med GNU/Linux och andra POSIX-system. Detta gäller i båda riktningarna, och gäller exempelvis också jämförelser mellan Windows och DOS. En användare av ordbehandlingsprogrammet Word Perfect för DOS tryckte exempelvis **Shift+F3** för att spara, och skulle kunna hävda att det är "ologiskt" att du ska trycka **Ctrl+s** i Windows. Detsamma gäller alla former av gnäll som bottnar i att användaren är besviken på att GNU/Linux inte *är* Windows. Att byta till GNU/Linux är att byta operativsystem, det är inte en uppgradering av Windows till en nyare version. Detsamma skulle gälla i andra riktningen.

Windows är inget standardiserat operativsystem utan en produkt från Microsoft. Notera dock att grundläggande POSIX-funktionalitet faktiskt finns implementerad i Windows från och med Windows NT, dock inte som standard utan i form av tilläggspaketet "Windows services for UNIX". Anledningen till detta är antagligen att vissa företag och myndigheter uppställt en policy som säger att operativsystem som används måste följa denna standard. Många anser dock att denna implementation är så bristfällig att den inte går att använda, t.ex. FAQ:n för nyhetsgruppen för Win32-programmering[19]. I senare versioner av Windows Services for UNIX har stödet för POSIX blivit bättre.

Utöver detta kan nämnas att Microsoft även utvecklat och marknadsfört ett eget renodlat POSIX-operativsystem för Intel-familjen med namnet Xenix. Det är lögn att påstå att Microsoft skulle ha något allvarligt emot konceptet med POSIX och UNIX som sådant — i själva verket uppfattade Microsoft Xenix som sin framtida huvudprodukt, och det var först Apple Lisa och Apple Macintosh-datorerna som fick dem att istället satsa stora resurser på att utveckla ett "grafiskt skal" ovanpå det dåvarande dominerande operativsystemet DOS. Detta kom med tiden att ombildas till ett eget operativsystem i och med Windows 95, som var den första variant av Windows som slutligen släppte beroendet av DOS.

A.1 Affärsmodellen och kundperspektivet

Jag har redan i kapitel 3 på sidan 111 beskrivit en del egenskaper som kännetecknar fri mjukvara. Vad kännetecknar då proprietär mjukvara?

Proprietär mjukvara utvecklas alltid primärt i vinstsyfte. Detta utesluter inte att utvecklarna finner andra nöjen än pengar i sitt arbete, men sådana prioriteringar är för programvaruföretaget alltid av underordnad betydelse. Många utvecklare av fri mjukvara, i synnerhet distributörer som Red Hat eller Novell, är också primärt drivna av vinst-

intresse. När del gäller stora delar av programvaruprojekten som utgör en distribution: GNU, Linux och X.org-servern är det inte vinstintresset som är i centrum, utan teknik och gemenskap. Även detta beskrevs utförligt i avsnittet om fri mjukvara.

Proprietär mjukvara alltid en *produkt*. När du köper och använder produkten, anser du dig inte vara en del av utvecklarlaget bakom produkten, och du vill ofta ha det på det viset också. När det gäller fri mjukvara, kan du bara köpa den som en *tjänst*. Du köper alltså inte själva mjukvaran — denna är oftast möjlig att ladda ned från Internet — utan tjänsten att driftsätta och hjälp med att använda mjukvaran.

Om du *inte* köper fri mjukvara som en tjänst, skall du betrakta dig själv som *medlem av ett projekt*, inte som *kund*. Är du inte intresserad av att vara medlem av något projekt, bör du vända dig till något företag (Red Hat, Novell o.s.v.) och köpa fri mjukvara som en tjänst.

Båda dessa kategoriseringar gäller i dubbel mån om du är i färd att köpa in ett operativsystem till en organisation, men där är tjänsteperspektivet det normala synsättet, och således inte särskilt överraskande.

Som privatperson kommer du i princip att uppleva en kartong med en distribution av GNU/Linux som likvärdig med en kartong innehållande Microsoft Windows. Båda erbjuder ett visst användarstöd (engelska: support) via telefon och Internet.

I större organisationer som vill införa operativsystem efterlyses ofta *certifieringar*, vilket innebär att tekniker i en stödorganisation har utbildats av systemleverantören och godkänts som kompetent att sköta ett visst system. Microsoft har flera sådana certifieringar, men både Red Hat och Novell har likvärdiga certifieringsprogram.[1]

Varken proprietära eller fria mjukvaror erbjuder någon som helst garanti för att programmen skulle vara felfria. Datorprogram har alltid fel, och därför vågar *ingen* utlova sådan garanti. En tjänsteorganisation tecknar avtal för att sköta viss drift med viss driftsäkerhet, och att finnas till hands för användarstöd under vissa specifika tider. Detta skiljer sig inte alls mellan Windows och POSIX-system.

När det gäller Microsoft Windows specifikt, så har detta operativsystem så stor spridning, att tillverkare av hårdvaror och mjukvaror är måna om att dessa skall fungerar tillsammans med Windows. Det är därför oftast lättare att få nyköpt hårdvara att fungera tillsammans med Windows. Som GNU/Linux-användare bör du aktivt välja sådana hård- och mjukvaror som utger sig för att fungera med GNU/Linux, medan du som Windowsanvändare, tack vare systemets dominerande ställning, kan ta för givet att det hela kommer att fungera.

[1] Du skall inte efterlysa en person som är ”certifierad på Linux” eller liknande saker, certifiering är kopplat till distributioner.

När det gäller äldre hårdvara kan förhållandet ibland vara det motsatta: den kan vara lätt att använda i GNU/Linux, eftersom stöd för den är en del av ett fritt mjukvaruprojekt, medan företag som gått i konkurs sällan utvecklar drivrutiner för gammal hårdvara till nya Windowsversioner. Mycket hårdvara som fungerar problemfritt tillsammans med GNU/Linux eller Windows 3.11 fungerar inte alls med Windows 2000.

A.2 Viktiga olikheter

Några viktiga punkter som kan vålla huvudbry är skillnader som råder mellan *alla* POSIX-system och Microsoft Windows:

- Windows har ett *grafiskt paradigm* vilket innebär att alla verktyg för systemet är skrivna för att användas i det grafiska användargränssnittet, inklusive alla administrativa program. I GNU/Linux används mestadels kommandoradstolken till systemadministration, medan de flesta tillämpningsprogram för vardagsbruk är grafiska.

- Tillämpningsprogram som användarna är vana vid finns inte till GNU/Linux. Istället finns liknande program med andra namn. Om du läser igenom kapitel 11 kommer du antagligen fram till vilka program som motsvarar vilka. (I värsta fall kan behovet lösas genom att Windows emuleras på olika vis, se nedan.)

- Windows avskiljer filnamn med bakåtställt snedstreck: \ (engelska: backslash) medan POSIX-systemen använder ett vanligt snedstreck (engelska: slash): /.

- Windows filsystem gör inte skillnad på stora och små bokstäver då t.ex. ett filnamn anges. Det gör alla POSIX-system: i Windows spelar det ingen roll om du anger *C:\FOO.TXT* fast filen heter *C:\fOo.tXt*. I GNU/Linux är detta två olika positioner i filsystemets namnrymd.

- Windows använder s.k. *enhetsbokstäver*, t.ex. *C:*, *F:* o.s.v. för att beteckna en viss hårdvara. Detta bruk kommer från DOS, som i sin tur plockat det från operativsystemet CP/M. POSIX-filsystemen är helt abstrakta och skiljer inte på vilken enhet en fil ligger på, och använder inte enhetsbokstäver.

- Eftersom Windows inte är avsett att plockas sönder i komponenter kan det vara svårt att se skarvarna, medan de i GNU/Linux är

mycket tydliga. Det är tydlig skillnad mellan komponenten ”Linux”, ”GNU-verktygen”, ”fönstersystemet *X*”, och ”Skrivbordsmiljön”, medan dessa komponenters motsvarigheter i Windows inte alltid är så lätta att urskilja.

A.3 Skalet

GNU/Linux kan starta skal motsvarande DOS-fönster i Windows, för att hantera interaktiva användarkommandon. Dessa kan startas från det grafiska gränssnittet som i GNU/Linux är fönstersystemet *X*, men också från en terminal, vilket kan vara allt från en anslutning i en serieport till en anslutning över Internet från andra sidan jorden. De flesta GNU/Linux-system har också ett antal *konsoller* tillgängliga om du trycker **Ctrl+Alt+F1** till **Ctrl+Alt+F6**. Dessa är *inte i sig* skal, utan bara terminaler där ett skal *kommer att* startas om du loggar in med ett giltigt användarnamn.

Ett vanligt tankefel bland före detta DOS- och Windowsanvändare är att tro att skalet i GNU/Linux-system är detsamma som det DOS-skal som tidigare låg ”under” Windows, då t.ex. Windows *startades från* DOS. Förmodligen har detta att göra med en intuitiv vantolkning av innebörden i den text som matas ut på skärmen då Linuxkärnan startas, och som i själva verket bara är just utskrifter från kärnan; detta är *inte* fråga om utskrifter i något ”skal” — det finns ingenting ”under” Linuxkärnan.

När Linuxkärnan har startat, kan den däremot öppna en eller flera terminaler, där skal i sin tur kan startas, och kärnan ligger då alltså *under* skalen, inte tvärt om. Kärnan är i sig abstrakt och har inget utseende vare sig i Windows eller GNU/Linux — den ger sig bara tillkänna genom olika meddelanden.

Det är helt möjligt att konfigurera GNU/Linux så att det bara går att logga in i det grafiskt, via fönstersystemet X. Flera inbyggda GNU/Linux-system har inget skal alls. Således: GNU/Linux kan starta skal, men skalet startar inte GNU/Linux. Kommandoprompten i t.ex. Windows 2000 fungerar likadant. Kommande versioner av Windows skall enligt uppgift vara försedd med ett nytt, kraftigt förbättrat skal som går under arbetsnamnet *Windows Shell* (WSH).

A.4 Lika för lika

Här följer en rad ”platser” eller ”ting” som finns i de flesta operativsystem; deras plats i Windows respektive deras plats i GNU/Linux.

- Lagringsplats för systeminställningar

  Windows: förr i *C:\CONFIG.SYS*, sedan *C:\Windows\WIN.INI* men numera i Windows Registry, som kan bearbetas med både *kontrollpanelen* och direkt med programmet *regedit.exe*. Registryn ligger i sin tur spridd över filerna SYSTEM.DAT och USER.DAT som ligger i *C:\Windows* i Windows 95/98, eller SAM, SECURITY, SOFTWARE, SYSTEM, DEFAULT och NTUSER.DAT som ligger i *C:\WINNT\System32\config* i Windows NT eller Windows 2000.

  GNU/Linux: i katalogen */etc* lagras alla systeminställningar i alla POSIX-system. Oftast i ren textform. */etc/passwd* och */etc/shadow* svarar mot Windows NT:s SAM-fil (Security Access Manager) (se avsnitt 2.3.4 på sidan 48).

- Lagringsplats för användarna

  Detta omfattar en lagringsplats för såväl sparade dokument, som inställningar och tillägg i operativsystemet som den enskilde användaren har gjort från sitt konto.

  Windows: Tidigare överallt på den primära hårddisken *C:*, sedan i *C:\My Documents* (eller motsvarande svenska *Mina Dokument*) eller *C:\Windows\Profiles\My Documents*, sedan till sist i *C:\Documents and Settings\Användarnamn*. Vissa nätverkssystem med Windows använder en nätverksbaserad *H:*-enhet för att användarna skall kunna lagra sina filer på en central filserver.[2] De flesta Windowsinställningar, t.ex. mappars synlighet och tillgång till systemfiler, måste göras på varje dator som användaren loggar in på. I Windows Registry lagras också användarinställningar (”profiles”) i USER.DAT eller NTUSER.DAT.

  GNU/Linux: alla användarens egna inställningar och dokument lagras i dennes hemkatalog. Inställningar för olika program lagras ofta i form av dolda filer, s.k. ”punktfiler”. Se vidare i avsnitt 2.3.5 på sidan 74 samt avsnitt 2.3.5 på sidan 73. Hemkatalogen kan lagras på en central filserver och delas mellan flera datorer på detta vis, eftersom POSIX-system kan *montera* en katalog vart som helst i filsystemet; t.ex. kan */home*, platsen där de flesta hemkataloger ligger, monteras på en NFS-baserad filserver på nätverket. På så vis följer hemkatalogen användaren, oavsett vilken dator denne loggar in

[2]Som i sin tur använder Microsoft-varianten av protokollet SMB.

på. USER.DAT eller NTUSER.DAT svarar mot inställningar i användarens hemkatalog i POSIX-systemen, men skrivbordssystemet GNOME har byggt en egen Registry-liknande komponent, som sedan i sin tur lagrar undan inställningarna i användarens hemkatalog.

- Uppstartfiler

  Detta innefattar startupskript och initiering; program som skall köras då datorn startas.

  Windows: förr i *C:\AUTOEXEC.BAT*, numera registernycklarna i HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run.

  GNU/Linux: programmet */sbin/init*. Detta anropar sedan olika varianter av startupskript, normalt i */etc/rcN.d* som i sin tur länkar till skript i */etc/init.d*. Se vidare avsnitt 5.4.3 på sidan 189.

- Bakgrundsprogram

  Detta betecknar automatiskt startade program som alltid skall vara igång.

  Windows: "panelhönor" i System Tray (små ikoner längst nere till höger i aktivitetsfältet), samt *Services* i kontrollpanelen. Startas antingen som en riktig servertjänst, eller via registerposten HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run

  GNU/Linux: *demoner* som startas antingen av *init*-processen vid systemuppstart via */etc/inittab* eller startup-skripen i */etc/init.d*. Se vidare avsnitt 2.3.3 på sidan 42 och avsnitt 5.4.3 på sidan 189. Utöver detta kan varje användare själv starta demonprogram och skrivbordssystemen gör det ofta möjligt att starta flera små tillägg automatiskt vid inloggning i det grafiska systemet. Dessa har i såväl GNOME som KDE stora likheter med Windows små panelikoner.

- Kärnan

  Detta beskriver vart operativsystemets kärna lagras och hur den startas.

  Windows: I DOS startas *C:\IO.SYS* och *C:\MSDOS.SYS*, senare (i Windows 95) enbart den första (MSDOS.SYS var då 0 byte stor). I Windows 3.11: de föregående samt *C:\Win.com* som i

sin tur laddar och kör *C:\WINDOWS\SYSTEM\krnl386.exe*. I Windows NT och Windows 2000: filen *C:\ntldr* som i sin tur laddar och kör *C:\NTDETECT.COM* som detekterar hårdvara, och sedan laddar och kör *C:\Winnt\System32\NTOSKRNL.EXE* som innehåller själva kärnan.

GNU/Linux: Här startas GRUB eller LILO på egen bootpartition (ligger i råa diskblock utanför filsystemet), som laddar och kör ett menysystem, som i sin tur kan ladda och köra både Linuxkärnor och olika Windows-varianter. Linuxkärnan ligger ofta i bootpartitionen med namnet *vmlinuz* och laddas in och startas av GRUB eller LILO.

- Dynamiska länkbibliotek

  Detta beskriver lagringsplatsen för dynamiska länkbibliotek (programbibliotek) som skall användas av hela systemet.

  Windows: består av DLL-filer (Dynamic Link Library) som ligger i *C:\Windows*, *C:\Windows\System* eller *C:\Windows\System32*. Versionshanteringen sköts genom att ett versionsnummer kodas in i DLL-filen.

  GNU/Linux: består av .so-filer som ligger i */lib*, */usr/lib*, eller */usr/local/lib*. Versionshanteringen sköts genom att .so-filen ges ett unikt suffix för varje ny version, t.ex. *libfoo.so.2*.

- Skriptfiler

  Detta beskriver systemet för skriptfiler med små interpreterade datorprogram för att utföra vissa automatiserade uppgifter i operativsystemet.

  Windows: tidigare .BAT-filer (batch, speciellt i DOS), numera .VBS (Visual Basic Script). Även perl används av många Windows-utvecklare. Skriptmotorn som kör VBS-skript kan plugga in olika moduler för att köra skript i alla möjliga språk.

  GNU/Linux: Bash-skript (se avsnitt 2.3.9 på sidan 90), Perlskript och Pythonskript. Alla kommandointerpretatorer kan användas, så även udda språk som awk eller TENEX C Shell förekommer.

- Utvecklingsmiljö

  Programutvecklingsverktyg för program till operativsystemet.

Windows: främst Microsoft Visual C++ och Microsoft Visual Basic. Även Borlands kompilatorer är populära. Samtliga måste köpas till separat.[3] Kompilatorn som används under Microsoft Visual C++ heter **cl.exe** och Make-verktyget som används heter **nmake**. Allt annat arbete för att förbereda kompilering sköts av Visual Studio självt.

GNU/Linux: GNU C Compiler, Make och GNU Autotools. Följer oftast med operativsystemet. Någon integrerad utvecklingsmiljö för editering och kompilering används oftast inte, istället används rena texteditorer som VIM och EMACS, därefter anropas GNU Make direkt för att bygga programmen. Om någon integrerad utvecklingsmiljö används så är det sannolikt Eclipse.

- Drivrutiner

  Detta beskriver systemet för att hantera små kodblock som används av operativsystemets kärna för att sköta hanteringen av viss hårdvara.

Windows: en stor databas med identifieringsinformation och drivrutiner som ofta utvecklats av hårdvaruleverantören medföljer Windows. Andra drivrutiner medföljer hårdvaran eller finns på leverantörens hemsida på Internet. Drivrutinerna levereras så gott som aldrig med källkod. Tillverkarna är måna om att se till att deras hårdvaror stödjs av Windows. Drivrutinerna ligger ofta som moduler i form av .SYS-filer[4] någonstans i *C:\Windows*-katalogen och kopplas till kärnan under körning. Till varje .SYS-fil svarar en .INF-fil som innehåller metadata om drivrutinen och talar om vilka enheter den är till för.

GNU/Linux: stora uppsättningar drivrutiner medföljer Linuxkärnan och X.org-servern. Somliga hårdvaror kräver patchning av kärnan och omkompilering. Hårdvaruleverantörerna kan ibland stödja Linuxprojektet med dokumentation och drivrutiner. Alla drivrutiner som medföljer systemet finns även som källkod. Vissa leverantörer tillhandahåller drivrutiner för egen hårdvara utan källkod. Tillverkarna har varierande intresse av att stödja GNU/Linux. Drivrutinerna ligger som

[3] GNU C Compiler finns också portad i ett par varianter som stödjer Microsoft Windows och som numera är ganska populära.

[4] Dessa .SYS-filer är i sin tur ett slags .DLL-filer.

moduler till kärnan eller X.org-servern, eller är helt inkompilerade som en del av kärnan.

- Processer

  Processer är det avtryck ett datorprogram gör i datorns minne då det körs: minne, filer som öppnas o.s.v.

  Windows: processer har fyra olika prioriteter (idle, normal, high och realtime) men användare förväntas inte manipulera processers prioriteter (det finns dock specialprogram som ändå kan göra det). Processer som kör i ett fokuserat fönster ges extra hög prioritet.

  GNU/Linux: processer har 39 olika prioriteter mellan -19 och 19. Avancerade användare förväntas kunna ändra prioriteter och döda processer som slutat fungera.

A.5 Windowsprogram i GNU/Linux

Den som byter från Windows till GNU/Linux har ofta utvecklat en del specialprogram, eller klarar sig bara inte utan ett eller annat Windowsprogram. Detta kan vara allt från reseräkningssystem till att hela Microsoft Office måste kunna användas. Det finns ett antal vanliga lösningar för den som behöver kunna köra Windowsprogram parallellt med GNU/Linux:

Dubbelbootsystem (engelska: *dual boot* innebär att både Windows och GNU/Linux ligger installerat på en och samma dator. Arbete kan dock inte utföras i båda systemen samtidigt, utan datorn måste startas om. Se vidare avsnitt 4.1.3 på sidan 138 där boot-förfarandet i GNU/Linux förklaras ingående.

VNC innebär att en separat Windows-dator sparas och används enbart via VNC-gränssnittet. Detta innebär att du öppnar ett fönster till den andra datorn och använder den på distans. Flera personer kan dela på denna dator men den kan bara användas av en användare åt gången. Se vidare om VNC i avsnitt 6.7 på sidan 265.

Även andra varianter på samma tema förekommer: *Citrix* säljer en proprietär mjukvara som ger möjlighet för flera användare att använda en Windowsmaskin via ett terminalprogram, som även finns till GNU/Linux och projektet *Rdesktop*[5] har skrivit ett program som gör att GNU/Linux-användare kan ansluta direkt till Windows Terminal Server som säljs av Microsoft.

[5] Se: `http://www.rdesktop.org`

QEMU [6] (Quick Emulator) är ett program som emulerar en hel IBM-kompatibel PC på låg maskinnära nivå. QEMU har även en utökning som stödjer *virtualisering*, vilket gör det möjligt för maskinkod för Intel-processorn att köras precis som vanlig kod, men så fort ett systemanrop sker skickar emulatorn anropet vidare till QEMU. Resultatet blir en snabb dator-i-datorn, på vilken vilket operativsystem som helst kan installeras. Det är möjligt att köra flera Windowsdatorer ”inuti” GNU/Linux, eller att köra flera GNU/Linux-system i egna QEMU-datorer. ”Datorn” lagras i en stor fil på hårddisken, ofta i användarens hemkatalog, och installation av ett operativsystem under QEMU kräver naturligtvis att du har tillgång till en kopia av det operativsystem du önskar köra. En proprietär mjukvara av samma slag som QEMU är **VMWare**, (Virtual Machine Software) som i vissa avseenden säkert är bättre men som inte tillför speciellt mycket mera prestanda.

Wine (akronym för *Wine is Not an Emulator*) är en fri mjukvara som implementerar Windows API, Win32, för GNU/Linux-system.[7] Windowsprogram som körs under Wine upplever driftsmiljön som om den vore Windows, fast den inte är det. Wine kan köra de flesta vanliga Windowsprogram med varierande resultat. Microsoft Office kan i princip alltid köras, och företaget Codeweavers säljer ett specialanpassat installationsprogram som kan användas för att installera Microsoft Office (Word, Excel, PowerPoint, Outlook, Internet Explorer) samt ett antal andra program (bl.a. Microsoft Visio) så att de kan köras direkt i GNU/Linux utan behov av någon Windowsinstallation. Ett annat företag, Transgaming Technologies, säljer en version av Wine som är speciellt inriktad mot att kunna hantera spel.[8] Även om Wine är instabilare än en emulator som QEMU har det betydligt högre prestanda och kräver inte att något Windows-operativsystem skall finnas tillgängligt för att programmet skall fungera. Wine skapar en konfigurationskatalog per användare där exempelvis ~/.wine/drive_c innehåller motsvarigheten till Windows C:\-enhet.

A.6 Samba — blandad miljö

Officiell hemsida: `http://www.samba.org/`

[6] Se: `http://fabrice.bellard.free.fr/qemu/`
[7] Se: `http://www.winehq.com`
[8] Se: `http://www.transgaming.com/`

Samba är det klassiska exemplet på hur GNU/Linux används i små och mellanstora företag. Programmet skapades julen 1991 då Andrew Tridgell med ledning av paket som skickades över ett nätverk mellan en Windowsklient[9] och en filserver med namnet *Pathworks*[10] började analysera protokollet som användes för att bygga en fristående Pathworksserver. Nätverksgränssnittet i Pathworks var en implementation av protokollet Server Message block (SMB), och detta råkade vara exakt samma protokoll som Microsoft själva använde för att dela filsystem över nätverk, men vid detta tillfälle hade Tridgell ingen aning om den saken.

Tidgell skrev snabbt ihop en enkel server som kunde hantera SMB, men först i slutet av 1992 började han köra GNU/Linux och blev då medveten om att SMB-servern även kunde kommunicera direkt med Microsoft Windows. Till en början kallades programmet *smbserver*, men en varumärkesdispyt gjorde att namnet ändrades till *Samba*, ett ord som ju innehåller bokstäverna S, M och B.

Samba gör det möjligt att dela ut resurser från ett GNU/Linux-system till flera Windows-datorer: såväl filservertjänster (vilket är det vanligaste) som primär och sekundär inloggningsdomän, skrivare[11] och Microsoft Active Directory kan hanteras med Samba. Tusentals Windowsanvändare kan alltså på detta vis få sina nätverkstjänster av en eller flera centrala servrar som inte kör Windows utan GNU/Linux och Samba.

Att konfigurera och starta Samba är inte särskilt enkelt, men väl värt besväret om du behöver skicka filer mellan GNU/Linux och Windowsdatorer på ditt hemnätverk, eller om du vill låta en Windowsdator använda din GNU/Linux-skrivare. Windows brukar använda en skrivardrivrutin installerad på varje Windowsmaskin, som sedan skickar utskriften i ett format som skrivaren förstår, via nätverket, Samba och CUPS, till skrivaren utan att bearbeta utskriften på vägen. Den kan också skicka den jobbet obearbetat, och sedan låta CUPS bearbeta utskriften med sina egna drivrutiner så att den passar skrivaren. Detta är lite smartare, eftersom det gör det möjligt att bygga en separat skrivarserver som vet allt om skrivaren, och låter klienten anropa denna för utskrifter, utan att klienten för den skull behöver veta något om skrivaren.

Det finns gott om dokumentation om såväl fil- som skrivardelning på Sambas hemsida för den som är hågad.

En vanlig missuppfattning om Samba är att det skulle vara användbart i andra riktningen: för att låta GNU/Linux-system kommunicera med Windowssystem. Så är det inte rent generellt. För att ansluta ett

[9] Windows 3.0 i detta fall.

[10] Pathworks användes av Windex, en *X*-server för Windows.

[11] Windowsdatorer kan även använda CUPS skrivarsystem direkt, via en speciell drivrutin som kan fås från CUPS hemsida. Då behövs inte Samba alls.

GNU/Linux-system till t.ex. en Windowsfilserver finns det SMB-stöd inbyggt i kärnan, som gör det möjligt att montera en utdelad katalog direkt över nätverket. Någon Sambainstallation behövs inte för detta. Emellertid underhålls även SMB-delen av Linuxkärnan av samma personer som arbetar med Samba.

Samba krävs däremot för att läsa av nätverket efter anslutna SMB-datorer,[12] för att veta vilka datorer som har utdelade enheter som det är möjligt att ansluta till. Detta är känt från Microsoft Windows som *Network Neighborhood* (nätverksgrannskapet). Såväl KDE som GNOME arbetar med att integrera stöd för detta i skrivbordsmiljöerna, och det fristående programmet *LinNeighborhood*[13] som bl.a. används av Mandrake Linux hjälper också till med detta.

Windowsskrivare kan också användas av GNU/Linux, men används då av programmet **smbprint** som kommer med Samba, men egentligen är ett fristående program. Det behövs ingen körande Sambademon för att få **smbprint** att fungera, så denna kan i förkommande fall stängas av.

[12]Detta görs med programmen **smbclient** och **nmblookup** som följer med Samba.
[13]Se: `http://www.bnro.de/~schmidjo/`

BILAGA B

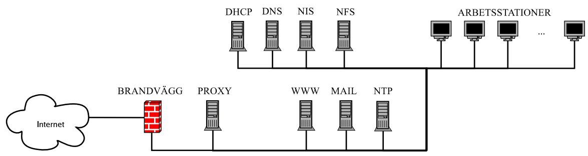
Stordrift av GNU/Linux-system

I detta appendix behandlas översiktligt stordrift av GNU/Linux-system. Att beskriva hur stordrift går till i någon form av detalj kräver en hel separat bok, förmodligen dessutom en bok i större format än denna. Ämnet behandlas därför här bara översiktligt, så att den intresserade kan förstå grunderna och själv gå vidare på rätt område för att lära sig mer.

Vad som menas med *stordrift* är för det första något flytande. Definitionen här är att det är ett nätverk av datorer, något större än ett hemnätverk, där det uppstått behov av att dela resurser. Med resurser menas hårddiskutrymme, skrivare, kontolistor, o.s.v. I takt med systemets storlek växer också säkerhetsbehovet. Den som driver ett sådant system är nästan alltid driftsansvarig för en organisation eller myndighet av något slag.

Med *system* menas i detta sammanhang flera datorer sammankopplade i ett lokalt, organisationsomfattande nätverk, ofta kallat för organisationens *intranät*, samt alla de allmänna tjänster som körs på olika servrar i detta nätverk. (Se figur B.1.) När en tekniker som sysslar med stordrift pratar om *systemet* är det alltså en mycket omfattande term som inbegriper väldigt mycket.

I övrigt är *support* är ett ord du måste förstå för att begripa de personer som sysslar med stordrift. När ett datorsystem drivs av en speciell



Figur B.1: En schematisk skiss över hur ett större system på ett lokalt nätverk, anslutet till Internet kan se ut. De olika servrarna tillhandahåller olika tjänster till klienterna — arbetsstationerna. De olika servrarna kan i praktiken vara samlade i en eller några fysiska datorer.

avdelning i en större organisation handlar många och långa diskussioner om hurvida saker stödjs av supporten, eller *supporteras* som den gängse språkvidriga branschtermen lyder. Detta avsnitt behandlar inte sådana saker, utan handlar mer om de övergripande principerna för hur ett stort GNU/Linux-baserat system skall byggas upp.

B.1 Katalogtjänster: YP, NIS, LDAP

Det första som brukar installeras på ett system bestående av flera datorer är en *katalogtjänst* (engelska: *directory service*), även kallat *autentiseringstjänst* (engelska: *authentication service*).

Syftet med denna tjänst är att lagra användarnas kontoinformation centralt, så att ett användarkonto inte måste läggas till på varje individuell dator genom att editera */etc/passwd*, */etc/shadow* och */etc/group*.

Istället för att systemet bara kontrollerar dessa tre filer för att kontrollera om en användare skall släppas in i systemet, komer det också att begära ut en användarlista från katalogservern. Ofta används flera parallella katalogservrar — det vore ju tråkigt om bara en server fanns, och denna skulle stanna. Då skulle plötsligt inga användare släppas in i systemet.

Katalogtjänster är oftast komplementära: de gäller *utöver* de ordinarie filerna i */etc*, och det går alltså fortfarande även att använda lokala konton i systemet om så önskas.

Det finns tre huvudsakliga katalogtjänster:

YP stod för *Yellow Pages* men detta namn var varumärkesskyddat av den amerikanska motsvarigheten till Gula Sidorna, och verktyget fick därför byta namn till **NIS**, *Network Information Service*. Detta var den första katalogtjänst som introducerades för POSIX-system, utvecklad av Sun Microsystems.

Du kan undersöka YP/NIS motsvarigheter till konto- och gruppfilerna med **ypcat passwd** eller **ypcat group** på ett system som använder NIS.

NIS användes ibland även som ett alternativ till DNS för att ge namn till datorer i nätverket, och kallas då inte katalogtjänst utan *namntjänst*. (engelska: *name service*)[1]

NIS+ är en förbättrad variant av NIS, främst i säkerhetshänseende. Det anses vara svårt att använda och stödet i GNU/Linux är inte att räkna med, så undvik detta om du inte absolut måste använda det.

NYS är en förbättrad variant av NIS, som nog är den vanligaste formen av katalogtjänst på nya installationer idag. Det hanterar bland annat */etc/shadow* på ett elegant vis.

LDAP *Lightweight Directory Access Protocol* är den branschstandard för katalogtjänster som successivt har utkristalliserats från ITU-T-standarden X.500. Eftersom en mängd komplexitet skalats av från X.500 kallas det just "lättviktskatalogtjänst". IETF har standardiserat LDAP version 3 (vilket är den version som bör användas) i form av RFC 3377. Den vanligaste servern för LDAP på GNU/Linux-system är OpenLDAP.[2]

LDAP används oftast tillsammans med Kerberos (RFC 1510) som har till uppgift att se till att den som försöker ansluta sig till en katalogtjänst verkligen är den som denne utger sig för att vara. Till denna verifikation används kryptografiska nycklar och signaturer.

Alla dessa katalogtjänster stödjs av nyare versioner av GNU C Library, genom ett generellt gränssnitt för olika katlogtjänster, något som kallas *Name Service Switch* (NSS) och som kan konfigureras i filen */etc/nsswitch.conf*. Det finns därutöver ett konkurrerande projekt med samma syfte, som kallas *Pluggable Authentication Module* (PAM).

Beroende på vilken katalogtjänst som väljs kan ytterligare konfiguration krävas, t.ex. skall NIS-server anges i */etc/yp.conf* för NIS.[3]

Det anses att LDAP med Kerberos är framtidens melodi vad beträffar katalogtjänster, och NIS-varianterna är i högsta grad ett osäkert och

[1] När det gäller namngivning av datorer finns idag tre stora system: DNS (vanligast), WINS (Windows namntjänst, stödjs även av Samba, se föregående appendix) och NIS.

[2] Se `http://www.openldap.org/`

[3] Se NIS-HOWTO för detaljer om NIS: `http://www.tldp.org/HOWTO/NIS-HOWTO/`

dåligt alternativ. Emellertid är de gamla varianterna utbredda, och eftersom det är ganska svårt att byta namntjänst i ett stort system — ungefär i stil med hjärt-lung-byte — är det många som fortfarande använder NIS. Detta har också lett till att kunskapen om hur LDAP och Kerberos skall användas inte är särskilt utbredd, och därmed är det svårt att hitta den hjälp som behövs för att få det hela att fungera.[4]

B.2 NFS

Den näst vanligaste formen av tjänst som behövs i ett stort system är en filserver av något slag. Filserverns uppgift är att lagra filer som skall vara gemensamma för hela systemet, d.v.s. för alla datorer i nätverket.

I POSIX-system är NFS, *Network File System* det vanligaste systemet för filservrar. NFS uppfanns ursprungligen av Sun Microsystems men är numera definerat i RFC 1813 (version 3) och RFC 3010 (version 4).

Det är lätt att göra GNU/Linux-datorer till NFS-servrar: se till att NFS-demonen **nfsd** körs i ditt system, och rada upp katalogerna som skall delas ut i filen */etc/exports*. I denna fil kan du även begränsa vilka datorer som får montera NFS-katalogerna.

NFS-demonen öppnar och stänger filer, samt ser till så att filer inte "krockar" på grund av att flera användare samtidigt läser från och skriver till dem.

Klienterna (arbetsstationerna) monterar sedan dessa exporterade kataloger på vanligt vis, så som beskrivs på sidan 214.

Vanligtvis brukar administratörer nöja sig med att montera */home*-katalogen, så att alla hemkataloger ligger på en filserver.[5] På det viset går ingen information förlorad om hårddisken i någon användares dator skulle gå sönder. Detta ställer naturligtvis krav på administratören att göra säkerhetskopior av alla hårddiskar på filservern, samt att dessa hårddiskar är mycket snabba (vilket ofta åstadkoms med RAID, se nedan), samt att nätverket mellan filservern och användarnas datorer är snabbt.

I vissa system väljer administratörerna att även montera filträdet */usr/local* på en NFS-server. Detta gör att om program installeras och uppgraderas i */usr/local* på filservern, så finns det genast tillgängligt på

[4]Turbo Fredriksson (jo han heter faktiskt så!) har skrivit ett omfattande HOWTO-dokument som finns tillgängligt via `http://www.bayour.com/LDAPv3-HOWTO.html`

[5]I praktiken kan det vara svårt att få plats med alla hemkataloger i en enda katalog på filservern, och därför ligger användarkatalogerna ofta en nivå ner i kataloghierarkin på filservern, t.ex. */home/a/foo*, */home/a/bar* o.s.v. så att bokstaven *a* i det här fallet betecknar en hårddisk. Sedan kan nästa hårddisk kallas *b* o.s.v.

alla datorer i nätverket. Omvänt gäller att om administratörerna råkar förstöra ett program eller bryta något beroende kan ett program sluta fungera för alla användare.

Version 2.6 av Linuxkärnan har stöd för NFS version 4 som har en väsentligt utökad säkerhet — bland annat använder det TCP istället för UDP för överföringen, och kan dessutom använda Kerberos för att verifiera den som ansluter sig till filservern. NFS version 4 betraktas dock som ”experimentell” och bör kanske inte användas i större system än på ett tag.

Andra och ofta mer väldesignade nätverksfilsystem, såsom Andrew File System (AFS) förekommer, men i flertalet organisationer är det i princip bara två system som används i drift: NFS (i olika versioner) och Microsofts SMB (som ju också implementeras av Samba).

AFS, Coda och andra avancerade filsystem är potentiellt mycket bättre än både NFS och SMB, men används knappt eftersom tekniken inte är särskilt beprövad, och eftersom ingen vill vara först med att riskera sina livsviktiga nätverkssystem genom att använda något obeprövat, så växer användandet av dessa alternativa system mycket långsamt.

B.3 RAID

Även RAID, *Redundant Array of Independent Disks* eller *Redundant Array of Inexpensive Disks* är populärt vid drift av större POSIX-system av alla slag. RAID innebär att flera hårddiskar ansluts parallellt, och oftast att data lagras på mer än ett ställe.

RAID används av olika skäl. För det första kan flera parallellkopplade diskar öka hastigheten vid skrivning eller läsning till diskarna.

För det andra kan mycket stora lagringsutrymmen skapas genom att parallellkoppla många billiga diskar för att åstadkomma ett gemensamt stort lagringsutrymme. Det enklaste sättet att åstadkomma mycket billigt lagringsutrymme är att ta samtliga hårddiskleverantörers produktkataloger, räkna fram vilken hårddisk som ger mest megabytes per krona, köpa ett antal sådana och koppla samman dem som en RAID-disk. För operativsystemet uppträder en RAID-disk som om den vore en enda stor hårddisk.

För det tredje kan RAID sättas upp för att vara feltolerant: om en hårddisk i en sammankopplad RAID-disk går sönder (vilket inträffar förr eller senare) kan den bytas ut, och den data som fanns på denna hårddisk kan sedan återskapas från de övriga hårddiskarna, utan att någon data går förlorad. Det är detta som avses när hårddiskarna säges

vara *redundanta*: det finns mer lagringsutrymme än vad som faktiskt utnyttjas, och det extra lagringsutrymmet används för att lagra information som gör att en trasig disk kan återskapas. I professionella system kan detta dessutom oftast göras under drift, s.k. *hot swap*.

Det finns tre huvudsakliga varianter av RAID:

RAID 0 är bara ett annat namn för mycket enkelt parallellkopplade diskar. Om du har 8 hårddiskar med 6 gigabyte lagringsutrymme så har du efter sammankoppling med RAID 0 en 48 gigagbyte stor hårddisk. Denna hårddisk har också högre överföringshastighet, och det är egentligen det vanligaste skälet till att RAID 0 används: det är framför allt populärt vid ljud- och videoredigering då stora mediafiler lagras på hårddiskarna. En direkt fara med RAID 0 är att det räcker med att *en* av diskarna går sönder för att *hela* filsystemet skall gå sönder.

RAID 1 är den enklaste formen av redundans: diskarna sitter två och två, och det som skrivs till den ena disken skrivs också till den andra. Om sedan den ena disken går sönder, finns all information kvar på den andra. Detta ger *mycket* hög feltolerans, eftersom det är mycket ovanligt att två diskar går sönder samtidigt, men är samtidigt ganska dyrt, eftersom det dubblerar priset för lagringsutrymmet.

RAID 3, 4 och 5 bygger på att paritetsbitar lagras undan någonstans på en av diskarna. Om t.ex. fyra diskar används kommer resultatet att bli att tre av dem används för att lagra data, och den fjärde lagrar en kontrollsumma, som gör att informationen på vilken som helst av de fyra diskarna kan återskapas om *en* och *endast en* av de fyra diskarna går sönder. Detta görs genom modulo-2-addition (exklusive OR) av varje individuell bit på de fyra diskarna. Skillnaden mellan RAID 3 och RAID 4 och 5 är att de två senare tillämpar olika principer för att sprida ut checksumman över flera diskar, istället för att låta en enda disk ta hand om alla checksummor, vilket RAID 3 gör.

RAID 6 är en version av RAID som introducerar tillräckligt mycket redundans för att klara av det fall då två diskar går sönder samtidigt.

Det finns också RAID 2 men detta används inte mycket. Linuxkärnan stödjer inte RAID 2 och 3, men däremot RAID 0, 1, 4, 5 och 6. (RAID 6 är ännu experimentell.)

B.4 Internetservrar

Om du behöver tillhandahålla någon av de Internettjänster som presenteras i kapitel 9 behöver du sätta upp servrar för dem, d.v.s. i praktiken en eller flera demoner som svarar på anrop på vissa TCP- eller UDP-portar. Var och en av dessa demoner svarar naturligtvis mot ett speciellt program, och i detta avsnitt listas protokoll och motsvarande demon för ett antal välkända Internetprotokoll.

Vissa av servrarna är fristående, medan andra startas indirekt via internetdemonen **xinetd**.

DNS sköts av en namnserver, och hanterar namnsystemet för en eller flera domäner. Det populäraste programmet för att sköta DNS är **bind** — Berkeley Internet Name Domain, men även **DJBDNS** används en del.

HTTP sköts av en webbserver. Den första webbserver som skrevs av Tim Berners-Lee hade naturligt nog namnet **httpd**, liksom NCSA:s klon *NCSA HTTPd* som sedermera slutat underhållas. En modifierad version av NCSA:s programvara lever dock kvar under namnet *Apache*,[6] och är idag den i branschen dominerande programvaran för webbservertjänster.

Apache brukar ofta användas tillsammans med den svenska databasen MySQL och skripspråket PHP och formar då den så kallade ”PHP-triaden”. Om triaden dessutom körs under GNU/Linux kallas summan ibland för LAMP: Linux-Apache-MySQL-PHP.

SMTP sköts av en *epostserver* (engelska: *mail server*, kallas även *mail transfer agent*, MTA) och det vanligaste programmet för detta är **sendmail**, ett program som är i rakt nedstigande led släkt med ARPANET-programmet **delivermail** och har rykte om sig att vara notoriskt svårkonfigurerat. Enklare program som också ibland anses vara bättre är **qmail**, **exim** och **postfix**.

På sidan 388 redogjordes för epostsituationen i de flesta GNU/Linux-system som används idag: nio av tio användare kör inte en epostserver, men många program tror ändå att en sådan alltid finns tillgänglig.

Epostservern används både för att leverera epost till användare på det egna systemet och för att leverera epost till användare på andra system. Till det lokala systemet levereras det antingen direkt till en fil med användarens namn i katalogen */var/spool/mail* eller via ett separat brevlevereringsprogram som t.ex. **procmail**.

[6] Se: `http://httpd.apache.org/`

FTP sköts av en FTP-server (något annat namn används så vitt jag vet inte). FTP skall helst inte användas annat än till anonym tillgång till stora filer, men till just detta fungerar FTP mycket bra. Populära serverprogram är **ftpd** som medföljer GNU Inetnutils, **wu-ftpd** eller **vsftpd**.

IMAP *Internet Message Access Protocol* är en speciell server som körs på en mailserver för att tillhandahålla brevlådor åt användare. IMAP-servern körs parallellt med SMTP-servern: de två fyller helt olika ändamål. IMAP är ett system för att användare skall kunna lagra och hämta sin post, medan SMTP är något användarna använder för att skicka iväg sin post.

IMAP tillåter att användarna lagrar stora filer, "mappar" med epost i sina hemkataloger på systemet där det körs om de så vill. Användarna kan även skapa nya epostmappar vid behov. Det vanligaste är dock att användarna använder IMAP för att tömma sin brevlåda i */var/spool/mail*.

En annan vanlig typ av Internetserver är en *proxy* (betyder ungefär "fullmakt") för HTTP- och FTP-trafik. Denna har till uppgift att övervaka varje begäran om dokument från Internet från det lokala nätverket. Alla begärda dokument mellanlagras i proxyservern. Den stora vinsten med detta är när många användare under kort tid vill komma åt samma webbsida eller fil: den kopia som hämtats av den första användaren har mellanlagrats och skickas sedan direkt till alla andra användare som senare begär samma fil. Ett vanligt proxy-program är *Squid Cache*, men även Apache kan användas som proxyserver.

B.5 Driftövervakning

När du väl har ett stort system med flera viktiga servrar körande är det inte speciellt roligt om någon av dem plötsligt skulle stanna eller sluta fungera. Stora datorsystem placeras dessutom ofta i speciella maskinrum som ofta ligger ganska otillgängligt.

Av denna anledning används driftövervakningsprogram för stora system. Dessa program kör demoner på varje dator som skall övervakas, och kontrollerar med jämna mellanrum att allting fungerar som det skall. Övervakningen koordineras från en central övervakningsdator som sammanställer data om systemen och lagrar undan statistik över tillgänglighet och drifttider etc.

Det mest använda programmet för driftövervakning av GNU/Linux-

system är *Nagios*,[7] en efterföljare till det tidigare mest populära programmet *Net Saint*. Nagios övervakar såväl datorer som nätverk och fysisk driftsmiljö (om mätutrustning finns i driftlokalen).

Om något händer i ett system under drift kan Nagios självständigt vidta vissa åtgärder: starta om tjänster, skicka mail eller SMS till administratörer o.s.v. Det finns även ett gränssnitt för att själv tillverka insticksprogram för att övervaka speciella tjänster.

[7] Se: `http://www.nagios.org/`

Litteraturförteckning

[1] *A Brief History of the Internet*, The Internet Society
http://www.isoc.org/internet/history/brief.shtml
verifierad 2004-02-04

[2] **Andreasson, Oskar**: *Iptables Tutorial*
http://iptables-tutorial.frozentux.net/iptables-tutorial.html
verifierad 2004-02-27

[3] **Arnison, Matthew**: *The means to an X for Linux: an interview with David Dawes from XFree86.org*
http://www.cat.org.au/maffew/cat/xfree-dawes.html
verifierad 2004-01-15

[4] **Bailey, Edward C.**: *Maximum RPM*, Red Hat Inc., Durham, USA 2000, ISBN 1-888172-78-9

[5] **Bowden, Terrehon**, **Bauer, Bodo** och **Nerin, Jorge**: *The /proc filesystem*, textfil som finns i */Documentation/filesystems/proc.txt* i Linux-kärnans källkodsträd.

[6] **Braun, David**: *Disk Encryption HOWTO*,
http://www.tldp.org/HOWTO/Disk-Encryption-HOWTO/
verifierad 2004-02-29

[7] *comp.os-research FAQ*
http://www.faqs.org/faqs/os-research/
verifierad 2003-05-15

[8] *Debian Social Contract*
`http://www.debian.org/social_contract`
verifierad 2003-10-30

[9] **Ettrich, Matthias**, *New Project: Kool Desktop Environment. Programmers wanted!* Usenet-artikel, finns bl.a. på
`http://groups.google.com/groups?selm=53tkvv$b4j@newsserv.zdv.uni-tuebingen.de`
Verifierad 2004-01-14

[10] *Filesystem Hierarchy Standard*
`http://www.pathname.com/fhs/`
verifierad 2003-11-03

[11] **Free Software Foundation**, *The Free Software Definition*
`http://www.gnu.org/philosophy/free-sw.html`
verifierad 2003-11-14

[12] **Henning, Michi**, *Computing Fallacies [or: What is the World Coming To?]*
`http://mail.gnome.org/archives/gnome-hackers/2002-February/msg00072.html`
verifierad 2003-05-08

[13] **Icaza, Miguel de**: *Let's Make Unix Not Suck*
`http://primates.ximian.com/~miguel/bongo-bong.html`
verifierad 2003-07-27

[14] **IEEE Standard 1003.1-2001: Standard for Information Technology – Portable Operating System Interface (POSIX)**. Denna består i sin tur av fyra olika dokument: **Base Definitions**, **Rationale**, **Shell and Utilities** och **System Interfaces**.

[15] **Ilshammar, Lars** och **Larsmo, Ola**, *404 — Utflykter i glömskans landskap*, Atlas, Avesta 2005, ISBN 91-7389-189-4

[16] **Körner, Ulf**: *Köteori*, Studentlitteratur, Lund 2003, ISBN 91-44-02908-X

[17] **Levy, Steven**: *Hackers — Heroes of the Computer Revolution*, Dell Publishing, New York 1984, ISBN 0-385-31210-5

[18] *Linux Standard Base*, Free Standard Group
`http://www.linuxbase.org/spec/`
verifierad 2003-11-23

[19] **Loughran, Steve**, *Frequently Asked Questions About Win32 Programming*
`http://www.iseran.com/Win32/FAQ/faq.htm`
verifierad 2003-05-08

[20] **Love, Robert M.**, *Linux Kernel Development, Second Edition*, Novell Press, 2005, ISBN 0672327201

[21] **Moody, Glyn**, *Rebel Code — Linux and the Open Source Revolution*, ISBN 0-14-029804-5

[22] *Multiboot Specification*, Free Software Foundation
`http://www.gnu.org/manual/grub-0.92/html_mono/multiboot.html`
verifierad 2003-11-02

[23] **Olofsson, Jessica**, *Upphovsrättsliga aspekter på licenser för fri programvara och öppen källkod*, Institutet för Rättsinformatik, IRI-rapport 2003:1, ISSN 0281-1286

[24] *Open Sources*, O'Reilly & Associates Inc., ISBN 1-56-592582-3

[25] **Raymond, Eric S.**, *Katedralen och basaren* (svensk översättning av Sven Wilhelmsson)
`http://home.swipnet.se/swi/KatB-se.html`
verifierad 2004-01-16

[26] **Raymond, Eric S.**, *The Art of Unix Programming*, ISBN 0-13-142901-9

[27] *Red Hat Customization Guide*, Red Hat
`http://www.redhat.com/docs/manuals/linux/RHL-9-Manual/custom-guide/`
verifierad 2003-11-03

[28] **Rubini, Alessandro**, *Booting the Kernel*
`http://www.linux.it/~rubini/docs/boot/boot.html`
verifierad 2003-11-17

[29] **Salus, Peter**, *A Quarter Century of Unix*, ISBN 0201547775

[30] **Silberschatz, Abraham**, *Operating System Concepts, 6th Edition*, ISBN 0471417432

[31] **Single UNIX Specification**, från **Open Group** finns på `http://www.opengroup.org/` men kan vara svår att lokalisera exakt, version 3 av texten finns exempelvis på

`http://www.opengroup.org/onlinepubs/007904975/`
verfierad 2003-05-08

[32] **Stallman, Richard M.**, *New UNIX Implementation* Usenet-artikel, finns bl.a. på
`http://groups.google.com/groups?hl=en&selm=771@mit-eddie.UUCP`
verifierad 2003-11-10

[33] **Torvalds, Linus**, *Just for Fun*, Alfabeta Bokförlag AB, Stockholm 2002, ISBN 951-50-1284-8

[34] **Torvalds, Linus**, brev till *comp.os.minix* den 25 augusti 1991 meddelande-ID 1991Aug25.205708.9541@klaava.Helsinki.FI

[35] **Torvalds, Linus** och **Tanenbaum, Andy** förde år 1992 en tröttsam debatt om mikrokärnor versus monolitiska kärnor
`http://www.dina.dk/~abraham/Linus_vs_Tanenbaum.html`
verifierad 2003-05-20

[36] **Vaughan, Gary V.**, **Elliston, Ben**, **Tromey, Tom** och **Lance Taylor, Ian**: *GNU Autoconf, Automake and Libtool*, USA 2000, ISBN 1-57870-190-2

[37] **Walker, John Mark**, *There Is No Open Source Community*
`http://www.onlamp.com/pub/a/onlamp/2006/01/12/no_oss_community.html`
verifierad 2006-01-21

[38] **Walleij, Linus**, *Copyright finns inte*, Rootgear, Kungälv 2000, ISBN 91-630-9465

[39] **Williams, Sam**, *Free As In Freedom — Richard Stallman's Crusade For Free Software*, O'Reilly & Associates, Inc. Sebastopol, Kanada, 2002, ISBN 0-596-00287-4

Sakregister